

SciencesPo Computational Economics

Spring 2017

Florian Oswald

April 6, 2017

1 Computational Economics: Constrained Optimization

Florian Oswald Sciences Po, 2017

1.1 Constrained Optimisation

- Recall our generic definition of an optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } \begin{array}{ll} c_i(x) = 0, & i \in E \\ c_i(x) \geq 0, & i \in I \end{array}$$

- E is the set of *equality constraints* and I is the set of *inequality constraints*.
- Definition: The Feasible Set:** Let Ω be the set of points x that satisfy the constraints, i.e.

$$\Omega = \{x \mid c_i(x) = 0, i \in E; c_i(x) \geq 0, i \in I\}$$

- Then, a different way of writing our problem is

$$\min_{x \in \Omega} f(x)$$

- A vector x^* is a *local solution* to this problem if $x^* \in \Omega$ and there is a neighborhood \mathcal{N} s.t.

$$f(x) \geq f(x^*), \forall x \in \mathcal{N} \cap \Omega$$

- Definition: The Active Set:** Active set $\mathcal{A}(x)$ at any feasible x consists of the equality constraint indices from E together with the indices of the inequality constraints for i for which $c_i(x) = 0$; that is,

$$\mathcal{A}(x) = E \cup \{i \in I \mid c_i(x) = 0\}$$

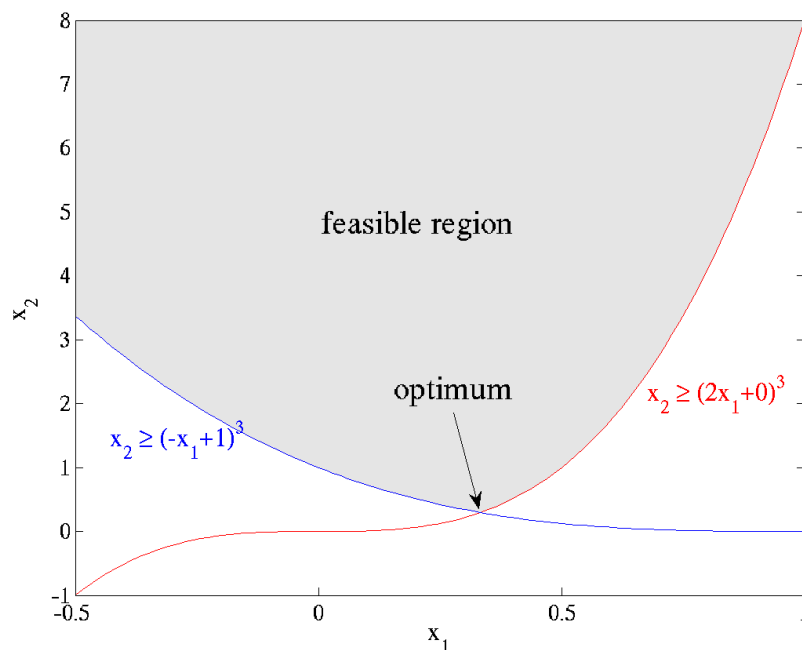
- At a feasible point x , the inequality constraint $i \in I$ is said to be *active* if $c_i(x) = 0$, and *inactive* if $c_i(x) > 0$

1.2 Nonlinear Constraints

- Consider the following problem

$$\min_{x \in \mathbb{R}^2} \sqrt{x_2} \text{ subject to } \begin{aligned} x_2 &\geq 0 \\ x_2 &\geq (a_1 x_1 + b_1)^3 \\ x_2 &\geq (a_2 x_1 + b_2)^3 \end{aligned}$$

- This configuration of constraints leads to the following feasible region for parameters $a_1 = 2, b_1 = 0, a_2 = -1, b_2 = 1$.



1.3 Example: 1 equality constraint

- consider

$$\min x_1 + x_2 \quad s.t. \quad x_1^2 + x_2^2 - 2 = 0$$

- constraint is a circle with radius $\sqrt{2}$ centered at 0. The solution must lie *on* that circle.
- Solution: $(-1, -1)$. Consider any other point on circle, like $(\sqrt{2}, 0)$.

1.4 Example: 1 inequality constraint

- Let's modify this example to

$$\min x_1 + x_2 \text{ such that } 2 - x_1^2 - x_2^2 \geq 0$$

- constraint is the region inside a circle with radius $\sqrt{2}$ centered at 0. The solution must lie *on or inside* that circle.

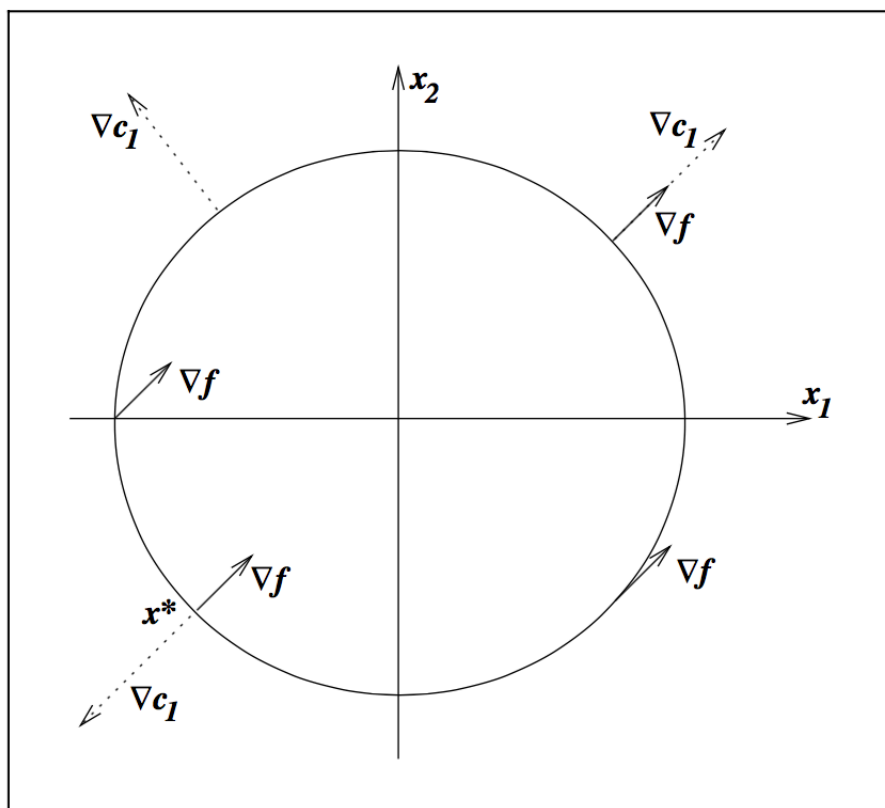


Figure 12.3 in [nocedal-wright][2]

- Solution: $(-1, -1)$. Consider any other point on circle, like $(\sqrt{2}, 0)$.
- Two cases:
 1. x lies strictly inside the circle, and $c_1(x) > 0$
 2. x lies strictly on the circle, and $c_1(x) = 0$
- Complementarity condition.

1.5 Some Methods

- Penalty Function and Augmented Lagrangian Methods
- Sequential Quadratic Method
- Interior Point Method

1.6 Constrained Optimisation with `NLopt.jl`

- We need to specify one function for each objective and constraint.
- Both of those functions need to compute the function value (i.e. objective or constraint) *and* it's respective gradient.
- Notice that we can disregard $x_2 \geq 0$ here.
- NLopt expects constraints **always** to be formulated in the format

$$g(x) \leq 0$$

where g is your constraint function

- The constraint function is formulated for each constraint at x . it returns a number (the value of the constraint at x), and it fills out the gradient vector, which is the partial derivative of the current constraint wrt x .
- There is also the option to have vector valued constraints, see the documentation.
- We set this up as follows:

```
In [1]: function myfunc(x::Vector, grad::Vector)
    if length(grad) > 0
        grad[1] = 0
        grad[2] = 0.5/sqrt(x[2])
    end
    return sqrt(x[2])
end

function constraint(x::Vector, grad::Vector, a, b)
    if length(grad) > 0
        # modifies grad in place
        grad[1] = 3a * (a*x[1] + b)^2
        grad[2] = -1
    end
    return (a*x[1] + b)^3 - x[2]
end

using NLopt
# define an Opt object: which algorithm, how many dims of choice
opt = Opt(:LD_MMA, 2)
```

```

# set bounds and tolerance
lower_bounds!(opt, [-Inf, 0.])
xtol_rel!(opt, 1e-4)

# define objective function
min_objective!(opt, myfunc)
# define constraints
# notice the anonymous function
inequality_constraint!(opt, (x,g) -> constraint(x,g,2,0), 1e-8)
inequality_constraint!(opt, (x,g) -> constraint(x,g,-1,1), 1e-8)

#call optimize
(minf,minx,ret) = optimize(opt, [1.234, 5.678])

```

Out [1]: (0.5443310477213124, [0.333333, 0.296296], :XTOL_REACHED)

1.7 NLOpt: Rosenbrock

- Let's tackle the rosenbrock example again.
- To make it more interesting, let's add an inequality constraint.

$$\min_{x \in \mathbb{R}^2} (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \text{ subject to } 0.8 - x_1^2 - x_2^2 \geq 0$$

- in NLOpt format, the constraint is $x_1 + x_2 - 0.8 \leq 0$

```

In [2]: function rosenbrock(x::Vector, grad::Vector)
    if length(grad) > 0
        grad[1] = -2.0 * (1.0 - x[1]) - 400.0 * (x[2] - x[1]^2) * x[1]
        grad[2] = 200.0 * (x[2] - x[1]^2)
    end
    return (1.0 - x[1])^2 + 100.0 * (x[2] - x[1]^2)^2
end
function r_constraint(x::Vector, grad::Vector)
    if length(grad) > 0
        grad[1] = 2*x[1]
        grad[2] = 2*x[2]
    end
    return x[1]^2 + x[2]^2 - 0.8
end
opt = Opt(:LD_MMA, 2)
lower_bounds!(opt, [-5, -5.0])
min_objective!(opt, (x,g) -> rosenbrock(x,g))
inequality_constraint!(opt, (x,g) -> r_constraint(x,g))
ftol_rel!(opt, 1e-9)
(minf,minx,ret) = optimize(opt, [-1.0, 0.0])

```

Out [2]: (0.07588358473630112, [0.724702, 0.524221], :FTOL_REACHED)

1.8 JuMP.jl

- Introduce [JuMP.jl](#)
- JuMP is a mathematical programming interface for Julia. It is like AMPL, but for free and with a decent programming language.
- The main highlights are:
 - It uses automatic differentiation to compute derivatives from your expression.
 - It supplies this information, as well as the sparsity structure of the Hessian to your preferred solver.
 - It decouples your problem completely from the type of solver you are using. This is great, since you don't have to worry about different solvers having different interfaces.
 - In order to achieve this, JuMP uses [MathProgBase.jl](#), which converts your problem formulation into a standard representation of an optimization problem.
- Let's look at the readme
- The technical citation is Lubin et al [1]

1.9 JuMP: Quick start guide

- this is from the [quick start guide](#)
- please check the docs, they are excellent.

1.9.1 Step 1: create a model

- A model collects variables, objective function and constraints.
- it defines a solver to be used.

```
using Clp
m = Model(solver=ClpSolver()) # provide a solver

# Define variables
@variable(m, x ) # No bounds
@variable(m, x >= lb ) # Lower bound only (note: 'lb <= x' is not valid)
@variable(m, x <= ub ) # Upper bound only
@variable(m, lb <= x <= ub ) # Lower and upper bounds

# we can create arrays of a variable
N = 2
@variable(m, x[1:M,1:N] >= 0 )

# or put them in a block
@variables m begin
    x
    y >= 0
    Z[1:10], Bin
    X[1:3,1:3], SDP
    q[i=1:2], (lowerbound = i, start = 2i, upperbound = 3i)
    t[j=1:3], (Int, start = j)
```

```
end
```

```
# Equivalent to:
@variable(m, x)
@variable(m, y >= 0)
@variable(m, Z[1:10], Bin)
@variable(m, X[1:3,1:3], SDP)
@variable(m, q[i=1:2], lowerbound = i, start = 2i, upperbound = 3i)
@variable(m, t[j=1:3], Int, start = j)

# bounds can depend on indices
@variable(m, x[i=1:10] >= i )
```

1.10 Objective and Constraints

- We can easily add objective and constraint functions:

```
@constraint(m, x[i] - s[i] <= 0) # Other options: == and >=
@constraint(m, sum(x[i] for i=1:numLocation) == 1)
@objective(m, Max, 5x + 22y + (x+y)/2) # or Min
```

- This is fully integrated with Julia. you can use the generator syntax for sums:

```
@objective(sum(x[i] + y[i]/pi for i = I1, j = I2 if i+j < some_val))
```

```
In [3]: ## Simple example
```

```
using JuMP
using Clp

m = Model(solver = ClpSolver())
@variable(m, 0 <= x <= 2 )
@variable(m, 0 <= y <= 30 )

@objective(m, Max, 5x + 3*y )
@constraint(m, 1x + 5y <= 3.0 )

print(m)

status = solve(m)

println("Objective value: ", getobjectivevalue(m))
println("x = ", getvalue(x))
println("y = ", getvalue(y))
```

```
Max 5 x + 3 y
Subject to
  x + 5 y  3
  0  x  2
  0  y  30
```

```
Objective value: 10.6
x = 2.0
y = 0.2
```

```
In [4]: # JuMP: Rosenbrock Example
        # Instead of hand-coding first and second derivatives, you only have to give `JuMP` expressions
        # Here is an example.
```

```
using JuMP
using Ipopt

let

    m = Model(solver=IpoptSolver())

    @variable(m, x)
    @variable(m, y)

    @NLobjective(m, Min, (1-x)^2 + 100(y-x^2)^2)

    solve(m)

    println("x = ", getvalue(x), " y = ", getvalue(y))

end
```

```
*****
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 3.12.4, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).
```

```
Number of nonzeros in equality constraint Jacobian...:      0
Number of nonzeros in inequality constraint Jacobian.:      0
Number of nonzeros in Lagrangian Hessian...:              3

Total number of variables...:      2
      variables with only lower bounds:      0
      variables with lower and upper bounds:      0
      variables with only upper bounds:      0
Total number of equality constraints...:      0
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  1.0000000e+00  0.00e+00  2.00e+00 -1.0  0.00e+00   -  0.00e+00  0.00e+00  0
  1  9.5312500e-01  0.00e+00  1.25e+01 -1.0  1.00e+00   -  1.00e+00  2.50e-01f  3
  2  4.8320569e-01  0.00e+00  1.01e+00 -1.0  9.03e-02   -  1.00e+00  1.00e+00f  1
  3  4.5708829e-01  0.00e+00  9.53e+00 -1.0  4.29e-01   -  1.00e+00  5.00e-01f  2
  4  1.8894205e-01  0.00e+00  4.15e-01 -1.0  9.51e-02   -  1.00e+00  1.00e+00f  1
  5  1.3918726e-01  0.00e+00  6.51e+00 -1.7  3.49e-01   -  1.00e+00  5.00e-01f  2
  6  5.4940990e-02  0.00e+00  4.51e-01 -1.7  9.29e-02   -  1.00e+00  1.00e+00f  1
  7  2.9144630e-02  0.00e+00  2.27e+00 -1.7  2.49e-01   -  1.00e+00  5.00e-01f  2
  8  9.8586451e-03  0.00e+00  1.15e+00 -1.7  1.10e-01   -  1.00e+00  1.00e+00f  1
  9  2.3237475e-03  0.00e+00  1.00e+00 -1.7  1.00e-01   -  1.00e+00  1.00e+00f  1
iter   objective   inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr ls
 10  2.3797236e-04  0.00e+00  2.19e-01 -1.7  5.09e-02   -  1.00e+00  1.00e+00f  1
 11  4.9267371e-06  0.00e+00  5.95e-02 -1.7  2.53e-02   -  1.00e+00  1.00e+00f  1
 12  2.8189505e-09  0.00e+00  8.31e-04 -2.5  3.20e-03   -  1.00e+00  1.00e+00f  1
 13  1.0095040e-15  0.00e+00  8.68e-07 -5.7  9.78e-05   -  1.00e+00  1.00e+00f  1
 14  1.3288608e-28  0.00e+00  2.02e-13 -8.6  4.65e-08   -  1.00e+00  1.00e+00f  1

```

Number of Iterations...: 14

```

                                (scaled)                (unscaled)
Objective...:  1.3288608467480825e-28  1.3288608467480825e-28
Dual infeasibility...:  2.0183854587685121e-13  2.0183854587685121e-13
Constraint violation...:  0.0000000000000000e+00  0.0000000000000000e+00
Complementarity...:  0.0000000000000000e+00  0.0000000000000000e+00
Overall NLP error...:  2.0183854587685121e-13  2.0183854587685121e-13

```

```

Number of objective function evaluations      = 36
Number of objective gradient evaluations      = 15
Number of equality constraint evaluations      = 0
Number of inequality constraint evaluations    = 0
Number of equality constraint Jacobian evaluations = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations     = 14
Total CPU secs in IPOPT (w/o function evaluations) = 0.118
Total CPU secs in NLP function evaluations    = 0.029

```

EXIT: Optimal Solution Found.

x = 0.9999999999999899 y = 0.9999999999999792

```

In [5]: # not bad, right?
        # adding the constraint from before:

```

```

let

    m = Model(solver=IpoptSolver())

    @variable(m, x)
    @variable(m, y)

    @NLobjective(m, Min, (1-x)^2 + 100(y-x^2)^2)
    @NLconstraint(m, x^2 + y^2 <= 0.8)

    solve(m)

    println("x = ", getvalue(x), " y = ", getvalue(y))

end

```

This is Ipopt version 3.12.4, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

```

Number of nonzeros in equality constraint Jacobian...:      0
Number of nonzeros in inequality constraint Jacobian.:      2
Number of nonzeros in Lagrangian Hessian...:              5

Total number of variables...:      2
      variables with only lower bounds:      0
      variables with lower and upper bounds:    0
      variables with only upper bounds:      0
Total number of equality constraints...:      0
Total number of inequality constraints...:      1
      inequality constraints with only lower bounds:      0
      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  1.0000000e+00  0.00e+00  2.00e+00  -1.0  0.00e+00   -  0.00e+00  0.00e+00  0
  1  9.5312500e-01  0.00e+00  1.25e+01  -1.0  5.00e-01   -  1.00e+00  5.00e-01f  2
  2  4.9204994e-01  0.00e+00  9.72e-01  -1.0  8.71e-02   -  1.00e+00  1.00e+00f  1
  3  2.0451702e+00  0.00e+00  3.69e+01  -1.7  3.80e-01   -  1.00e+00  1.00e+00H  1
  4  1.0409466e-01  0.00e+00  3.10e-01  -1.7  1.46e-01   -  1.00e+00  1.00e+00f  1
  5  8.5804626e-02  0.00e+00  2.71e-01  -1.7  9.98e-02   -  1.00e+00  1.00e+00h  1
  6  9.4244879e-02  0.00e+00  6.24e-02  -1.7  3.74e-02   -  1.00e+00  1.00e+00h  1
  7  8.0582034e-02  0.00e+00  1.51e-01  -2.5  6.41e-02   -  1.00e+00  1.00e+00h  1
  8  7.8681242e-02  0.00e+00  2.12e-03  -2.5  1.12e-02   -  1.00e+00  1.00e+00h  1
  9  7.6095770e-02  0.00e+00  6.16e-03  -3.8  1.37e-02   -  1.00e+00  1.00e+00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr ls
 10  7.6033892e-02  0.00e+00  2.23e-06  -3.8  3.99e-04   -  1.00e+00  1.00e+00h  1
 11  7.5885642e-02  0.00e+00  2.07e-05  -5.7  7.99e-04   -  1.00e+00  1.00e+00h  1

```

```

12  7.5885428e-02  0.00e+00  2.74e-11  -5.7  1.38e-06      -  1.00e+00  1.00e+00h  1
13  7.5883585e-02  0.00e+00  3.19e-09   -8.6  9.93e-06      -  1.00e+00  1.00e+00f  1

```

Number of Iterations...: 13

	(scaled)	(unscaled)
Objective...:	7.5883585442440671e-02	7.5883585442440671e-02
Dual infeasibility...:	3.1949178858070582e-09	3.1949178858070582e-09
Constraint violation...:	0.0000000000000000e+00	0.0000000000000000e+00
Complementarity...:	2.5454985882932001e-09	2.5454985882932001e-09
Overall NLP error...:	3.1949178858070582e-09	3.1949178858070582e-09

Number of objective function evaluations	=	20
Number of objective gradient evaluations	=	14
Number of equality constraint evaluations	=	0
Number of inequality constraint evaluations	=	20
Number of equality constraint Jacobian evaluations	=	0
Number of inequality constraint Jacobian evaluations	=	14
Number of Lagrangian Hessian evaluations	=	13
Total CPU secs in IPOPT (w/o function evaluations)	=	0.004
Total CPU secs in NLP function evaluations	=	0.002

EXIT: Optimal Solution Found.

x = 0.7247018392092258 y = 0.5242206029480763

1.11 JuMP: Maximum Likelihood

- Let's redo the maximum likelihood example in JuMP.
- Let μ, σ^2 be the unknown mean and variance of a random sample generated from the normal distribution.
- Find the maximum likelihood estimator for those parameters!
- density:

$$f(x_i|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

- Likelihood Function

$$\begin{aligned}
L(\mu, \sigma^2) &= \prod_{i=1}^N f(x_i|\mu, \sigma^2) = \frac{1}{(\sigma\sqrt{2\pi})^n} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^N (x_i - \mu)^2\right) \\
&= (\sigma^2 2\pi)^{-\frac{n}{2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^N (x_i - \mu)^2\right)
\end{aligned}$$

- Constraints: $\mu \in \mathbb{R}, \sigma > 0$

- log-likelihood:

$$\log L = l = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^N (x_i - \mu)^2$$

- Let's do this in JuMP.

```
In [6]: # Copyright 2015, Iain Dunning, Joey Huchette, Miles Lubin, and contributors
# example modified
using JuMP
using Distributions

distrib = Normal(4.5,3.5)
n = 10000

data = rand(distrib,n);

m = Model(solver=IpoptSolver())

@variable(m, mu, start = 0.0)
@variable(m, sigma >= 0.0, start = 1.0)

@NLobjective(m, Max, -(n/2)*log(2*sigma^2)-sum((data[i] - mu) ^ 2 for i = 1:n)/(2*sigma^2))

solve(m)
println(" = ", getvalue(mu), ", mean(data) = ", mean(data))
println("^2 = ", getvalue(sigma)^2, ", var(data) = ", var(data))
```

This is Ipopt version 3.12.4, running with linear solver mumps.

NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

```
Number of nonzeros in equality constraint Jacobian...:      0
Number of nonzeros in inequality constraint Jacobian.:      0
Number of nonzeros in Lagrangian Hessian...:           3
```

```
Total number of variables...:          2
      variables with only lower bounds:      1
      variables with lower and upper bounds:  0
      variables with only upper bounds:      0
```

```
Total number of equality constraints...:      0
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  1.7211927e+05  0.00e+00  1.01e+02  -1.0  0.00e+00   -  0.00e+00  0.00e+00   0
  1  1.2147584e+05  0.00e+00  9.58e+01  -1.0  9.27e+00   -  1.00e+00  5.00e-01f  2
```

2	7.4758242e+04	0.00e+00	3.97e+01	-1.0	2.37e-01	-	8.94e-01	1.00e+00f	1
3	4.9624139e+04	0.00e+00	1.64e+01	-1.0	3.08e-01	-	1.00e+00	1.00e+00f	1
4	3.6644573e+04	0.00e+00	6.64e+00	-1.0	3.88e-01	-	1.00e+00	1.00e+00f	1
5	3.0384758e+04	0.00e+00	2.60e+00	-1.0	4.69e-01	-	1.00e+00	1.00e+00f	1
6	2.7756981e+04	0.00e+00	9.40e-01	-1.0	5.20e-01	-	1.00e+00	1.00e+00f	1
7	2.6938155e+04	0.00e+00	2.73e-01	-1.7	4.72e-01	-	1.00e+00	1.00e+00f	1
8	2.6791555e+04	0.00e+00	5.45e-02	-1.7	3.06e-01	-	1.00e+00	1.00e+00f	1
9	2.6784961e+04	0.00e+00	2.83e-03	-2.5	8.44e-02	-	1.00e+00	1.00e+00f	1
iter	objective	inf_pr	inf_du	lg(mu)	d	lg(rg)	alpha_du	alpha_pr	ls
10	2.6784947e+04	0.00e+00	6.72e-06	-3.8	4.30e-03	-	1.00e+00	1.00e+00f	1
11	2.6784947e+04	0.00e+00	1.69e-09	-5.7	6.80e-05	-	1.00e+00	1.00e+00f	1
12	2.6784947e+04	0.00e+00	4.02e-13	-8.6	1.02e-06	-	1.00e+00	1.00e+00f	1

Number of Iterations...: 12

	(scaled)	(unscaled)
Objective...:	8.4800120957055221e+00	2.6784947040164145e+04
Dual infeasibility...:	4.0182611534353866e-13	1.2692070597731996e-09
Constraint violation...:	0.0000000000000000e+00	0.0000000000000000e+00
Complementarity...:	2.5064395506978949e-09	7.9168343001320250e-06
Overall NLP error...:	2.5064395506978949e-09	7.9168343001320250e-06

Number of objective function evaluations	= 18
Number of objective gradient evaluations	= 13
Number of equality constraint evaluations	= 0
Number of inequality constraint evaluations	= 0
Number of equality constraint Jacobian evaluations	= 0
Number of inequality constraint Jacobian evaluations	= 0
Number of Lagrangian Hessian evaluations	= 12
Total CPU secs in IPOPT (w/o function evaluations)	= 0.006
Total CPU secs in NLP function evaluations	= 0.023

EXIT: Optimal Solution Found.

= 4.490925092754868, mean(data) = 4.490925092754868
 $\hat{2}$ = 12.417569220495565, var(data) = 12.418811091779508

References

- [1] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *arXiv:1508.01982 [math.OC]*, 2015.
- [2] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.