Introduction to JuMP

Modeling Optimization Problems

Jordan Jalving

Department of Chemical and Biological Engineering
University of Wisconsin-Madison

UW-Madison
March 2018

Getting Started

- All code and slides are in GIT repo `https://github.com/jalving/JuMPTutorial_2018.git`
- We will use `Julia` notebooks that will be executed on `JuliaBox`
- Access `http://juliabox.com`
- Click on the git tab and clone GIT repo `https://github.com/jalving/JuMPTutorial_2018.git`
- Click on Jupyter tab and access notebooks under JuMPTutorial_2018

Why Optimize?

- Model real world physical systems
  - Thermodynamics
  - Micro-organism behavior
  - Parameter estimation
- Economics
  - Supply chain design
  - Production scheduling
  - Portfolio optimization
- Resiliency
  - Disaster recovery
  - Identify critical system failures
- Model Predictive Control
  - Robotics (e.g. Autonomous vehicles)
  - Market participation

Optimization Overview - Linear Programming (LP)

Standard Linear Programming Problem

minimize	$c^T x$	Objective function
subject to	$Ax = b$	Equality constraints
	$x \succeq 0$	Inequality constraints
	$x \in \mathbb{R}^n$	Continuous Variables

Example Linear Program

$$
\begin{aligned}
\text{minimize} \quad & x + y \\
\text{subject to} \quad & x + y \leq 1 \\
& x \geq 0, y \geq 0 \\
& x, y \in \mathbb{R}
\end{aligned}
$$

Algorithms:

- Simplex, Dual Simplex
- Interior Point methods
- LPs with millions of variables can be solved efficiently

Optimization Overview - Mixed Integer Linear Programming (MILP)

Standard Linear Programming Problem

$$
\begin{aligned}
\text{minimize} \quad & c^T x + d^T y \\
\text{subject to} \quad & Ax + By = f \\
& x \succeq 0, y \succeq 0 \\
& x \in \mathbb{R}^n, y \in \mathbb{Z}^p
\end{aligned}
$$

Example MILP

$$
\begin{aligned}
\text{minimize} \quad & x + y \\
\text{subject to} \quad & x + y \leq 1 \\
& x \geq 0 \\
& x \in \mathbb{R}, y \in \{0, 1\}
\end{aligned}
$$

Algorithms:

- Branch and Bound
- Branch and Cut
- Hueristic Methods

Optimization Overview - Nonlinear Programming (NLP)

Standard Nonlinear Programming (NLP) Problem

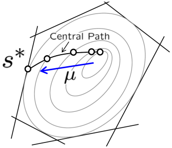
$$
\begin{aligned}
\min_{x}\ & f(x) && \text{Objective function (e.g., economics)} \\
s.t.\ \ & c(x) = 0 && \text{Equality constraints (e.g. physical equations)} \\
& g(x) \geq 0 && \text{Inequality constraints (e.g. physical limits)}
\end{aligned}
$$

Example NLP

$$
\begin{aligned}
\min_{x_1, x_2}\ & x_1^2 + x_2^2 \\
s.t.\ \ & x_1 + x_2 = 5 \\
& e^{x_1} \leq 4
\end{aligned}
$$

Ipopt - Interior Point OPTimizer

- To solve NLPs we will use the interior-point NLP solver `Ipopt`.
- `Ipopt` uses a logarithmic barrier to transform the NLP into an equality-constrained subproblem:



$$\min_x \phi^\mu(x) := f(x) - \mu \sum_{j=1}^{m} ln(g_j(x))$$

$$s.t.\ c(x) = 0$$

- The barrier subproblem is solved for decreasing values of μ to reach a solution of the original NLP. This is done by solving:

$$\begin{aligned} \nabla_x \mathcal{L}(x,\lambda) = \nabla_x \phi^\mu(x) + \nabla_x c(x)\lambda = 0 \\ c(x) = 0 \end{aligned} \implies \begin{bmatrix} H(x,\lambda) & \nabla_x c(x) \\ \nabla_x c(x)^T & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla_x \mathcal{L}(x,\lambda) \\ c(x) \end{bmatrix}$$

- Highly efficient sparse linear algebra and globalization strategies enable solution of complex NLPs.
- `Ipopt` can solve NLPs with *millions* of variables and constraints.

Julia and JuMP

`Julia`

- Language for high-performance scientific computing
- Extensive library of tools for optimization, statistics, plotting, graphs,...
- Performance comparable to C
- Designed to enable parallelism

`JuMP`

- Algebraic modeling language for optimization written in `Julia`
- Syntax close to natural mathematical expressions
- Processes problems at speeds comparable to `GAMS` and `AMPL`
- Automatic computation of derivatives (NLPs)

Key:

- JuMP is solver independent (supports all problem types).
- Compatible with open-source (e.g. Ipopt) and commercial (e.g. Gurobi) solvers.

Solving Optimization Problems with JuMP

Consider our previous NLP example:

$$\min_{x_1,x_2} \; x_1^2 + x_2^2$$
$$s.t. \; x_1 + x_2 = 5$$
$$e^{x_1} \leq 4$$

```
using JuMP
using Ipopt
m = Model()
@variable(m, x1)
@variable(m, x2)
@objective(m, Min, x1^2 + x2^2)
@constraint(m, x1+x2 == 5)
@NLconstraint(m, exp(x1) <= 4)
solve(m)
```

- `Ipopt` internally solves a sequence of barrier problems of the form:

$$\min_{x_1,x_2} \; x_1^2 + x_2^2 - \mu \log(4 - e^{x_1})$$
$$s.t. \; x_1 + x_2 = 5$$

- Example implemented in Julia notebook

`simple_nonlinear_model.ipynb`

Solving Optimization Problems with JuMP

`JuMP` also enables compact syntax expressions:

$$\min_{x_1, x_2} \sum_{j \in \{1,2\}} x_j^2$$
$$s.t. \sum_{j \in \{1,2\}} x_j = 5$$
$$e^{x_1} \leq 4$$

```julia
using JuMP
using Ipopt
m = Model()
S = collect(1:2)
@objective(m,Min,sum(x[j]^2 for j in S))
@variable(m,x[S])
@constraint(m,sum(x[j] for j in S) == 5)
@NLconstraint(m,exp(x[1]) <= 4)
solve(m)
```

Example implemented in Julia notebook

`simple_nonlinearmodel_set.ipynb`