

# Optimization in Julia

Madeleine Udell

Operations Research and Information Engineering  
Cornell University

Based on joint work with  
with Karanveer Mohan, David Zeng, Jenny Hong,  
Steven Diamond, and Stephen Boyd;  
Miles Lubin, Iain Dunning, and Joey Huchette.

ACC, May 23 2017

## What is an optimization problem?

optimization problem: nonlinear form

$$\begin{array}{ll}\text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \dots, m_1 \\ & h_i(x) = 0, \quad i = 1, \dots, m_2 \\ & x \in \mathcal{C}\end{array}$$

- ▶ objective  $f_0$
- ▶ inequality constraints  $f_i$
- ▶ equality constraints  $h_i$
- ▶ domain  $\mathcal{C}$

advantages:

- ▶ easy to formulate

## What is an optimization problem?

optimization problem: conic form

$$\begin{array}{ll}\text{minimize} & c^T x \\ \text{subject to} & b - Ax \in \mathcal{K},\end{array}$$

where  $\mathcal{K}$  is a **convex cone**:

$$x \in \mathcal{K} \iff rx \in \mathcal{K} \text{ for any } r > 0.$$

advantages:

- ▶ efficiently grok the structure of problem
- ▶ fast solvers

## Structure determines solvers

How should we solve this problem?

- ▶ LP solver?
- ▶ conic solver?
- ▶ nonlinear derivative based solver?
- ▶ operator splitting?

## Structure determines solvers

How should we solve this problem?

- ▶ LP solver?
- ▶ conic solver?
- ▶ nonlinear derivative based solver?
- ▶ operator splitting?

**What do we know about this problem's structure?**

# Structure

useful kinds of structure:

- ▶ is the problem convex?
  - ▶ is the objective convex?
  - ▶ is the domain convex?
  - ▶ are the inequality constraints convex?
  - ▶ are the equality constraints affine?
- ▶ is the problem representable in some standard form?
  - ▶ convex: LP, QP, SOCP, SDP, ...
  - ▶ nonconvex: MILP, MISOCP ...
- ▶ is the problem smooth?
- ▶ ...

## Optimization in Julia

model specifies structure; solvers exploit structure

- ▶ model (e.g., JuMP or Convex)
- ▶ glue (MathProgBase)
- ▶ solvers (e.g., GLKP, Gurobi, Mosek, ECOS, ...)

JuliaOpt curates all these solvers: <http://www.juliaopt.org/>

## Two major approaches

- ▶ JuMP: user specifies structure
- ▶ Convex: solver detects structure



# JuMP vs Convex

## JuMP

- ▶ lower level interface
- ▶ access to advanced solver features
- ▶ automatic differentiation
- ▶ support for conic and nonlinear programming

## Convex

- ▶ automatic structure detection
- ▶ automatic convexity proof
- ▶ can only solve convex problems

## JuMP: getting started

**demo:**

https:

`//github.com/JuliaSystems/ACC-2017/JuMP-intro.ipynb`

## JuMP features

- ▶ automatic differentiation
- ▶ solver callbacks

## JuMP extensions

- ▶ JuMPeR.jl: for robust optimization
- ▶ MultiJuMP.jl: for multi-objective optimization
- ▶ JuMPChance.jl: for probabilistic chance constraints
- ▶ StochDynamicProgramming.jl: for discrete-time stochastic optimal control problems
- ▶ PolyJuMP.jl: for polynomial optimization
- ▶ StructJuMP.jl: for block-structured optimization
- ▶ NLOptControl.jl: for formulating and solving nonlinear optimal control problems

## Convex.jl: detecting and exploiting structure

Convex.jl is an extensible framework for detecting and exploiting structure.

three kinds of structure (so far):

- ▶ convexity
- ▶ conic form
- ▶ multiconvexity

Induction detects; recursion exploits. Let's see how.

## Convex.jl in action

**demo:**

`https://github.com/JuliaSystems/ACC-2017/  
convex-intro.ipynb`

## Basic types

Expressions are defined inductively:

- ▶ A **variable** is an expression.

```
x = Variable(4)
```

- ▶ A **constant** is an expression.

```
y = [1,2,3,4]
```

- ▶ A **composite expression** is formed by applying a **function** to other expressions.

```
f = norm(x - y)^2 + sum(abs(x))
```

## Expressions: examples

(using prefix notation)

- ▶  $x + y \implies (+, (x, y))$
- ▶  $x[1] + x[2] \implies (+, ((\text{index}, (x, 1)), (\text{index}, (x, 2))))$
- ▶  $\log(x + 7y) \implies (\log, (+, (x, (*, (7, y)))))$

Every composite expression has a **head** (operation) and a (possibly empty) list of **children** (arguments).



## Basic types

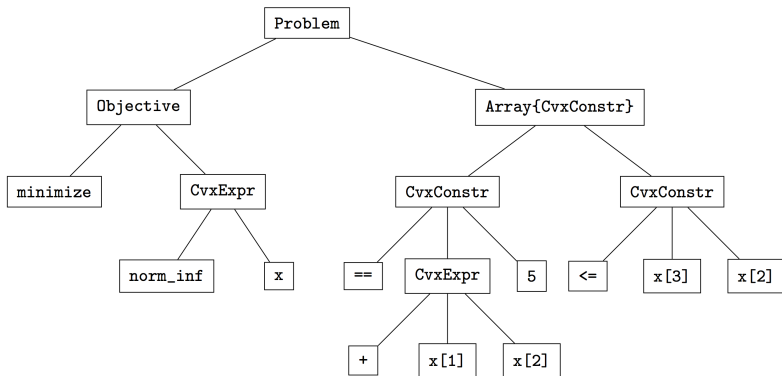
- ▶ A **constraint** is a comparison operator ( $\leq$ ,  $\geq$ ,  $=$ ,  $\preceq$ ,  $\succeq$ ) applied to two expressions.

`X <= 3 I`

- ▶ A **problem** is a sense (minimize, maximize, or satisfy) applied to an objective (any expression), along with a list of constraints.

```
problem = minimize(norm(x - y)^2,  
                    #= st =# x >= 1,  
                    sum(x) <= 10)
```

## Abstract expression tree for an optimization problem



## Structure by induction

We use induction (and recursion) to move from properties of

- ▶ variables,
- ▶ constants, and
- ▶ functions

to properties of

- ▶ expressions,
- ▶ constraints, and
- ▶ problems.

## Three case studies

- ▶ detect convexity
- ▶ transform to conic form
- ▶ detect multiconvexity

## Disciplined convex programming

**Disciplined convex programming** (DCP) [Grant, Boyd & Ye, 2006] provides a set of simple inductive rules to verify (but not falsify) convexity:

- ▶  $f \circ g(x)$  is convex in  $x$  if
  - ▶  $f$  is convex nondecreasing and  $g$  is convex
  - ▶  $f$  is convex nonincreasing and  $g$  is concave
- ▶  $f \circ g(x)$  is concave in  $x$  if
  - ▶  $f$  is concave nondecreasing and  $g$  is concave
  - ▶  $f$  is concave nonincreasing and  $g$  is convex

*cf.*, the chain rule:

$$(f \circ g)''(x) = f''(g(x))(g'(x))^2 + f'(g(x))g''(x)$$

A function is **DCP** if its convexity can be inferred from these composition rules.

## DCP: base case

A function *vexity* is defined on each data type (variable, constant, functions, constraints, problems) to return its **vexity**: constant, affine, convex, concave, or not DCP.

### **base case:**

- ▶ *Constant*. Constants are **constant**.
- ▶ *Variable*. Variables are **affine**.

## DCP: inductive rule

### inductive rules:

- ▶ *Expressions.* Functions each have known **curvature** (convex, concave, or affine) and **monotonicity** (increasing, decreasing, or none) in each of their arguments. Expressions check their convexity by examining convexity of arguments and following composition rules.
- ▶ *Constraints.* Constraints check their convexity by determining their left and right hand sides define convex sets.
- ▶ *Problems.* Problems check their convexity by verifying the objective and constraints are all convex.

## DCP: inductive rule

Composition rules are implemented as **arithmetic on vexities**:

$$\underbrace{\text{convex function}}_{\text{ConvexVexity}+} \underbrace{\text{nondecreasing}}_{\text{NonDecreasing}} \underbrace{\text{in}}_{*} \underbrace{\text{convex expression}}_{\text{ConvexVexity}} \underbrace{\text{is}}_{==} \underbrace{\text{convex}}_{\text{ConvexVexity}}$$

```
function vexity(x::AbstractExpr)
    monotonicities = monotonicity(x)
    vex = curvature(x)
    for i = 1:length(x.children)
        vex += monotonicities[i] * vexity(x.children[i])
    end
    return vex
end
```



## DCP expressions might as well be convex

Observe:

- ▶ if  $f$  is convex and nonincreasing and  $g$  is concave,
- ▶ then define  $\tilde{f}(x) = f(-x)$ ,  $\tilde{g}(x) = -g(x)$
- ▶ so

$$f(g(x)) = f(-(-g(x))) = \tilde{f}(\tilde{g}(x)),$$

$\tilde{f}$  is convex and nondecreasing and  $\tilde{g}$  is convex.

So let's suppose all functions are **convex** and *nondecreasing* in their arguments.

(This will simplify our exposition of conic form.)

## Conic form

A **conic form** optimization problem is written

$$\begin{array}{ll}\text{minimize} & c^T x \\ \text{subject to} & b - Ax \in \mathcal{K},\end{array}$$

where  $\mathcal{K}$  is a **convex cone**:

$$x \in \mathcal{K} \iff rx \in \mathcal{K} \text{ for any } r > 0.$$

examples:

- ▶ zero cone  $\mathcal{K} = \{0\}$
- ▶ positive orthant  $\mathcal{K} = \{x : x_i \geq 0, i = 1, \dots, n\}$
- ▶ second order cone  $\mathcal{K} = \{(x, t) : \|x\|_2 \leq t\}$
- ▶ positive semidefinite (PSD) cone  
 $\mathcal{K} = \{X : X = X^T, v^T X v \geq 0, \forall v \in \mathbf{R}^n\}$
- ▶ products of cones

## Conic form for expressions

epigraph conic form for expressions:

$$f(x) = \begin{array}{ll} \min & C[x; t] + d \\ \text{with variable} & t \\ \text{subject to} & A[x; t] + b \in \mathcal{K} \end{array}$$

(note: “objective” can be vector valued)

- ▶ function can be represented by tuple

$$(C, d, A, b, \mathcal{K})$$

## Conic form: base case

A function `conic_form` is defined on each data type (variable, constant, functions, constraints, problems) to return the tuple  $(C, d, A, b, \mathcal{K})$ .

**base case:**

► *Constant.*

$$\begin{array}{lll} & \min & 3 \\ 3 = & \text{with variable} & \emptyset \\ & \text{subject to} & \emptyset \end{array}$$

► *Variable.*

$$\begin{array}{lll} & \min & x \\ x = & \text{with variable} & \emptyset \\ & \text{subject to} & \emptyset \end{array}$$

## Conic form: inductive rule

**inductive rule:** if

$$f(y) = \begin{array}{ll} \min & C^f[y; t^f] + d^f \\ \text{with variable} & t^f \\ \text{subject to} & A^f[y; t^f] + b^f \in \mathcal{K}^f, \end{array}$$

$$g(x) = \begin{array}{ll} \min & C^g[x; t^g] + d^g \\ \text{with variable} & t^g \\ \text{subject to} & A^g[x; t^g] + b^g \in \mathcal{K}^g \end{array}$$

then

$$f(g(x)) = \begin{array}{ll} \min & C^f[C^g I][x; t^g; t^f] + C^f d^g + d^f \\ \text{with variable} & t^g, t^f \\ \text{subject to} & A^f[C^g I][x; t^g; t^f] + A^f d^g + b^f \in \mathcal{K}^f \\ & A^g[x; t^g] + b^g \in \mathcal{K}^g \end{array}$$

**proof:**  $f$  is convex and increasing in its argument and  $g$  is convex, so partial minimizations over  $t^f$  and  $t^g$  commute.

## Conic form: inductive rule

in math:

$$\begin{array}{ll} \min & C^f[C^g I][x; t^g; t^f] + C^f d^g + d^f \\ \text{with variable} & t^g, t^f \\ \text{subject to} & A^f[C^g I][x; t^g; t^f] + A^f d^g + b^f \in \mathcal{K}^f \\ & A^g[x; t^g] + b^g \in \mathcal{K}^g \end{array}$$

in code:

```
conic_form(f::AbstractExpr)
  (Cg,dg,Ag,bg,Kg) = conic_form(f.children)
  (Cf,df,Af,bf,Kf) = conic_form(f.head)
  return(Cf*[Cg I], Cf*dg+df, [Af*[Cg I]; Ag],
         [Af*dg + bf; bg], [Kf; Kg])
```

## Convex: rocket control

**demo:**

`https://github.com/Convex/examples/control.ipynb`

## JuMP: rocket control

**demo:**

```
https://github.com/JuliaSystems/ACC-2017/  
JuMP-Rocket.ipynb
```



## Multiconvex functions

### Definition (Restriction)

For  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  and  $\omega \subseteq \{1, \dots, n\}$ , define the **restriction**  $f_\omega(\cdot, \bar{x}) : \mathbf{R}^{|\omega|} \rightarrow \mathbf{R}$  of  $f$  to  $\omega$  to be the function obtained by fixing the coefficients in  $\omega^C$  to their values in  $\bar{x} \in \mathbf{R}^n$ :

$$x \mapsto f_\omega(x; \bar{x}).$$

## Multiconvex functions

### Definition (Restriction)

For  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  and  $\omega \subseteq \{1, \dots, n\}$ , define the **restriction**  $f_\omega(\cdot, \bar{x}) : \mathbf{R}^{|\omega|} \rightarrow \mathbf{R}$  of  $f$  to  $\omega$  to be the function obtained by fixing the coefficients in  $\omega^C$  to their values in  $\bar{x} \in \mathbf{R}^n$ :

$$x \mapsto f_\omega(x; \bar{x}).$$

### Definition (Multiconvex function)

A function  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  is  **$k$ -convex** if there exists a partition  $\Omega = \{\omega_1, \dots, \omega_k\}$  of  $\{1, \dots, n\}$  so that  $f_{\omega_j}$  is convex for every  $j = 1, \dots, k$ .

## Multiconvex functions

### Definition (Restriction)

For  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  and  $\omega \subseteq \{1, \dots, n\}$ , define the **restriction**  $f_\omega(\cdot, \bar{x}) : \mathbf{R}^{|\omega|} \rightarrow \mathbf{R}$  of  $f$  to  $\omega$  to be the function obtained by fixing the coefficients in  $\omega^C$  to their values in  $\bar{x} \in \mathbf{R}^n$ :

$$x \mapsto f_\omega(x; \bar{x}).$$

### Definition (Multiconvex function)

A function  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  is  **$k$ -convex** if there exists a partition  $\Omega = \{\omega_1, \dots, \omega_k\}$  of  $\{1, \dots, n\}$  so that  $f_{\omega_j}$  is convex for every  $j = 1, \dots, k$ .

Multiconvex functions generalize **biconvex** and **multilinear** functions.

- ▶ A 1-convex function is convex; a 2-convex function is biconvex; a 3-convex function is triconvex; etc.
- ▶ A multilinear function is multiconvex.

## Multiconvex problems

Consider a (nonconvex) optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x_{\beta_i}) \leq 0, \quad i = 1, \dots, m \end{array} \quad (\mathcal{P})$$

with variable  $x \in \mathbf{R}^n$ .

### Definition (Multiconvex problem)

An optimization problem is  **$k$ -convex** if there exists a partition  $\Omega = \{\omega_1, \dots, \omega_k\}$  of  $\{1, \dots, n\}$  with the following properties:

- ▶  $f_0$  is  $k$ -convex with partition  $\Omega$ ;
- ▶  $f_i$  is convex for every  $i = 1, \dots, m$ ;
- ▶ for every constraint  $i = 1, \dots, m$ , there is an element  $j$  of the partition with  $\beta_i \subseteq \omega_j$ .

## MultiConvex.jl

MultiConvex.jl extends Convex.jl to detect and (heuristically) solve multiconvex optimization problems using **disciplined** multiconvex programming:

- ▶ simple: less than 300 lines of code
- ▶ heuristic solution method: alternating minimization

## MultiConvex.jl

MultiConvex.jl extends Convex.jl to detect and (heuristically) solve multiconvex optimization problems using **disciplined** multiconvex programming:

- ▶ simple: less than 300 lines of code
- ▶ heuristic solution method: alternating minimization

### Definition (Disciplined multiconvex problem)

A multiconvex optimization problem is a **disciplined multiconvex problem** if

- ▶  $f_0$  is  $k$ -**convex** with partition  $\Omega = \{\omega_1, \dots, \omega_k\}$
- ▶  $f_0$  restricted to  $\omega_j$  is a disciplined convex function for every  $j = 1, \dots, k$
- ▶  $f_i$  is a disciplined convex function for  $i = 1, \dots, m$

## MultiConvex.jl in action

```
using MultiConvex

# initialize nonconvex problem
n, k = 10, 1
A = rand(n, k) * rand(k, n)
x = Variable(n, k)
y = Variable(k, n)
problem = minimize(sum_squares(A - x*y), x>=0, y>=0)

# perform alternating minimization on the problem
altmin!(problem)
```

## Conflict graphs

### Definition

The **conflict graph**  $G = (V, E)$  of a multiconvex expression  $e$  is a graph on the variables in the expression:

$$V = \mathbf{variablesin}(e), \quad E \subseteq V \times V$$

with the property that for any independent set of variables  $\omega$  in the graph, the restriction  $f_\omega$  of  $f$  to  $\omega$  is convex.

Every multiconvex expression has a (unique) conflict graph.



## Conflict graphs: recursion

- ▶ *Constant.* A constant  $c$  is multiconvex with conflict graph  $(\emptyset, \emptyset)$
- ▶ *Variable.* A variable  $v$  is multiconvex with conflict graph  $(v, \emptyset)$
- ▶ *Expressions.* The conflict graph of a composite expression is the union of the conflict graphs of its arguments, together with (possibly) a few more edges.
  - ▶ multiplication  $(*, (x, y))$  adds complete bipartite graph on **variablesin**( $x$ ) and **variablesin**( $y$ )
- ▶ *Constraints.* A constraint is multiconvex iff it is convex.
- ▶ *Problems.* Problems check their convexity by constructing a certifying partition  $\Omega$  of the conflict graph of the objective that respects the constraints (if one exists).

## Alternating minimization

Now that we've found a partition  $\Omega$ , we can use alternating minimization:

```
for iter=1:AMiters
    for  $\omega$  in  $\Omega$ 
        # free the variables in  $\omega$  to optimize over just those variables
        for v in  $\omega$ 
            free!(v)
        end
        solve!(problem, warmstart=true)
        # now that we've found their values, fix them again
        for v in  $\omega$ 
            fix!(v)
        end
    end
end
```

(or ADMM, or ...)

## Convex: wrapping up

Convex.jl is

- ▶ a modelling language that detects structure
  - ▶ (disciplined) convexity
  - ▶ conic form
- ▶ a framework for recursive reasoning about optimization problems
  - ▶ (disciplined) multiconvexity
  - ▶ easy to extend to detect new structures ...

More information (and code!)

- ▶ Convex.jl:  
<http://www.github.com/JuliaOpt/Convex.jl>
- ▶ MultiConvex.jl: <http://www.github.com/madeleineudell/MultiConvex.jl>
- ▶ Convex.jlpaper: <http://arxiv.org/abs/1410.4821>
- ▶ MultiConvex.jlpaper: <https://arxiv.org/abs/1609.03285>