The Type of Language for Mathematical Programming

Madeleine Udell

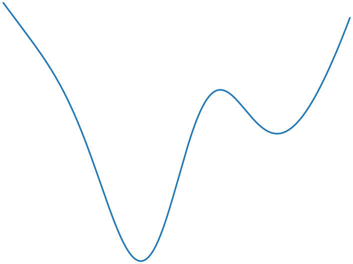
Operations Research and Information Engineering
Cornell University

JuliaCon, June 23 2017

What is an optimization problem?

the optimization contract:

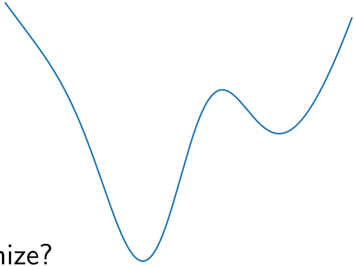
- you give me a function, and I'll find you its minimum



What is an optimization problem?

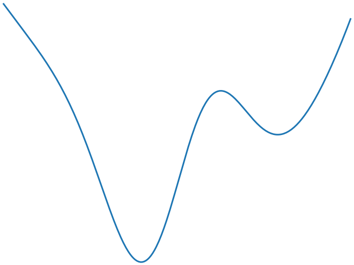
the optimization contract:

- you give me a function, and I'll find you its minimum



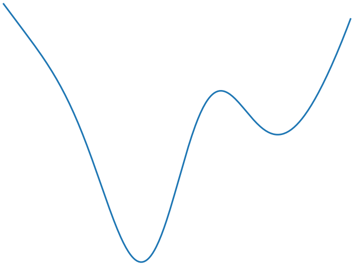
- why optimize?
  - fit a model to data (*e.g.*, understand customer preferences)
  - make predictions (*e.g.*, image recognition)
  - maximize revenue (*e.g.*, airline pricing)
  - maximize investment returns (*e.g.*, quant finance)
  - design a control system (*e.g.*, autopilot)
  - ...

How to find the minimum



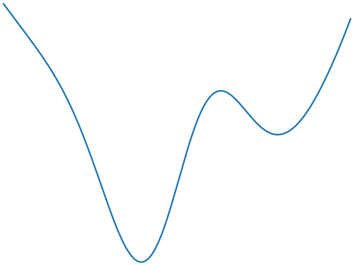
how to find the minimum of f?

How to find the minimum



how to find the minimum of f?

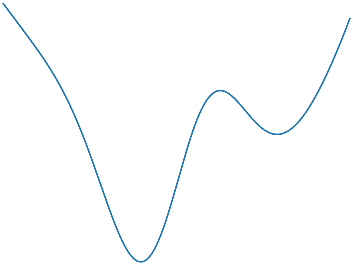
How to find the minimum



how to find the minimum of f?

- set derivative to zero

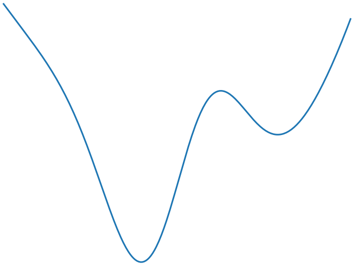
How to find the minimum



how to find the minimum of f?

- set derivative to zero ... if $\frac{d}{dx} f$ has an analytical formula

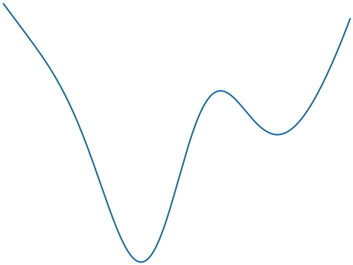
How to find the minimum



how to find the minimum of f?

- set derivative to zero . . . if $\frac{d}{dx}f$ has an analytical formula
- gradient descent / backprop

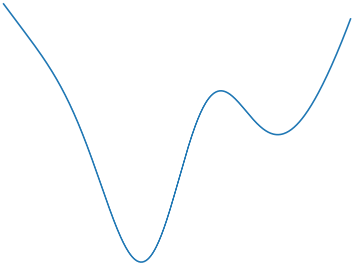
How to find the minimum



how to find the minimum of f?

- set derivative to zero . . . if $\frac{d}{dx} f$ has an analytical formula
- gradient descent / backprop . . . if f is differentiable

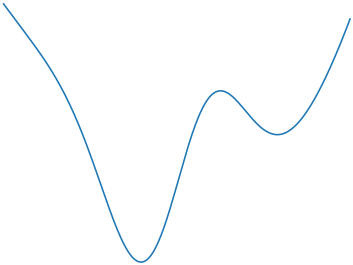
How to find the minimum



how to find the minimum of f?

- set derivative to zero ... if $\frac{d}{dx}f$ has an analytical formula
- gradient descent / backprop ... if f is differentiable
- other fancy optimization methods

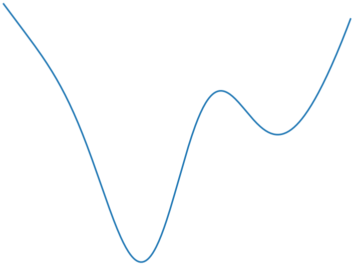
How to find the minimum



how to find the minimum of f?

- set derivative to zero ... if $\frac{d}{dx} f$ has an analytical formula
- gradient descent / backprop ... if f is differentiable
- other fancy optimization methods ... where they apply

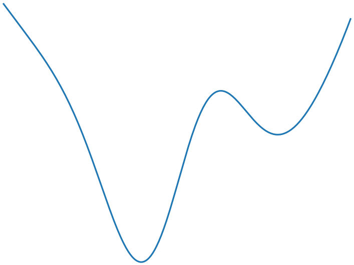
How to find the minimum



how to find the minimum of f?

- set derivative to zero ... if $\frac{d}{dx}f$ has an analytical formula
- gradient descent / backprop ... if f is differentiable
- other fancy optimization methods ... where they apply
- point

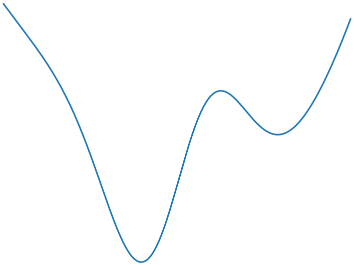
How to find the minimum



how to find the minimum of f?

- set derivative to zero . . . if $\frac{d}{dx}f$ has an analytical formula
- gradient descent / backprop . . . if f is differentiable
- other fancy optimization methods . . . where they apply
- point . . . if I can plot f in 2D or 3D

How to find the minimum



how to find the minimum of f?

- set derivative to zero ... if $\frac{d}{dx}f$ has an analytical formula
- gradient descent / backprop ... if f is differentiable
- other fancy optimization methods ... where they apply
- point ... if I can plot f in 2D or 3D

key question: how will you give me the function?

Example: what is $\frac{d}{dx}$?

what is

$$\frac{d}{dx}(x^2)|_{x=1}$$

?

Example: what is $\frac{d}{dx}$?

what is

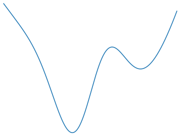
$$\frac{d}{dx}(x^2)|_{x=1}$$

? it's *bad notation!*

x means three different things above:

- x^2 is the square function
- $\frac{d}{dx}$ is an operator that takes a function and returns its derivative (another function)
- $|_{x=1}$ evaluates the function at the argument 1

How to give a function



- as a plot
- as an oracle

```
f(x) = x^2    › f
f(1)    1
f(2)    4
```

- as a type

```
type Square
end
f = Square()    Square()
evaluate(f::Square,x) = x^2
evaluate(f,1)    1
evaluate(f,2)    4
```

How to give a function

demo:

`https://github.com/madeleineudell/JuliaCon17/types-for-opt.ipynb`

How to give a function

moral:

- a function is a type
- on which various operations are defined
- which can be used to solve optimization problems

advantages:

- easy to understand
- easy to reuse code
- easy to extend by adding new methods

What is an optimization problem?

optimization problem: nonlinear form

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

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

advantages:

- easy to formulate

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

Convex in action

demo:

`https://github.com/madeleineudell/JuliaCon17/Convex-intro.ipynb`

Convex: behind the scenes

Convex is a framework for detecting and exploiting structure in optimization problems.

what properties of functions does Convex use?

- evaluate
- verify convexity
- compute conic form

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

Expressions: behind the scenes

(using prefix notation)

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

Every composite expression has

- a **head** (operation) and
- a (possibly empty) list of **children** (arguments).

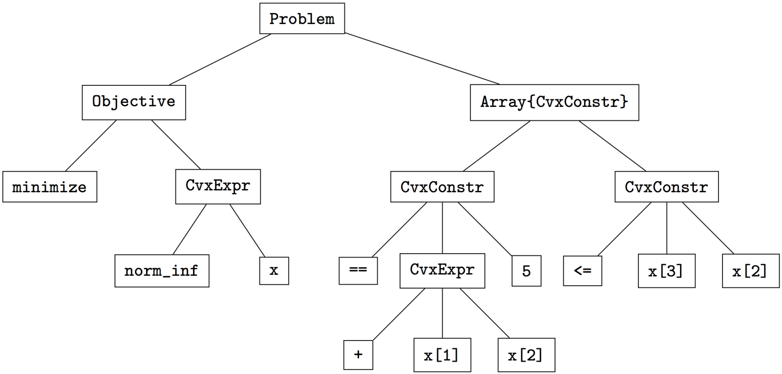
Evaluate expressions recursively

to evaluate expression:

- apply top level function to value of argument
- *e.g.*, if top level function of expression is abs,

```
function evaluate(e::AbsAtom)
  return abs.(evaluate(e.children[1]))
end
```

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.

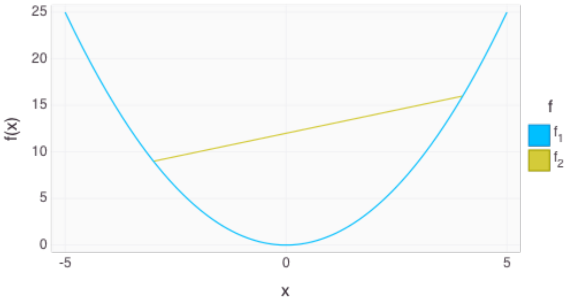
Detecting structure: two case studies

- detect convexity
- transform to conic form

Convexity

f is *convex* if for all $\theta \in [0,1]$

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$$



equivalently,

- f has nonnegative (upward) curvature
- the graph of f never lies above its chords

Disciplined convex programming

Disciplined convex programming (DCP) [Grant, Boyd & Ye, 2006] provides a set of simple inductive rules to verify 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

  cf., the chain rule:

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

Disciplined convex programming

Disciplined convex programming (DCP) [Grant, Boyd & Ye, 2006] provides a set of simple inductive rules to verify 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

  cf., the chain rule:

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

- $f \circ g(x)$ is concave in x if $-f \circ g(x)$ is convex
- $f \circ g(x)$ is affine if it is both convex and concave

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

Disciplined convex programming

Disciplined convex programming (DCP) [Grant, Boyd & Ye, 2006] provides a set of simple inductive rules to verify 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

  cf., the chain rule:

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

- $f \circ g(x)$ is concave in x if $-f \circ g(x)$ is convex
- $f \circ g(x)$ is affine if it is both convex and concave

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

(**N.B.** a function that is not DCP may still be convex)

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**:

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

```julia
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 in action

demo:

`https://github.com/madeleineudell/JuliaCon17/Convex-intro.ipynb`

Conic form

Convex transforms optimization problems to 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**:

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

examples:

- zero cone $\mathcal{K} = \{0\}$
- positive orthant $\mathcal{K} = \{x : x_i >= 0,\ i = 1, \ldots, n\}$
- 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

Why conic form?

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

advantages:

- efficiently grok the structure of problem
- fast solvers

Conic form for expressions

epigraph conic form for expressions:

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

(note: "objective" can be vector valued)

- function can be represented by tuple

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

Conic form: base case

A function `conic_form` is defined on each data type to return the tuple $(C, A, \mathcal{K})$.

base case:

- *Constant.*

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

- *Variable.*

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

Conic form: inductive rule

inductive rule: if

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

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

then

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

Conic form: inductive rule

inductive rule: if

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

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

then

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

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:

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

in code:

```
function conic_form(f::AbstractExpr)
  (Cg,Ag,Kg) = conic_form(f.children)
  (Cf,Af,Kf) = conic_form(f.head)
  return (Cf*[Cg I], [Af*[Cg I], [Ag 0]], [Kf, Kf])
end
```

Coda: compilers

is Convex reproducing the compiler?

- yes: and we're not ashamed
  - type system + multiple dispatch makes it easy
  - so you can simulate a compiler
  - without understanding Julia's own compiler

other optimization software goes down the rabbit hole:

- Automatic Differentiation **demo:**

  `https://github.com/madeleineudell/JuliaCon17/types-for-opt.ipynb`

The Type of Language for Mathematical Programming

Julia is the right language for mathematical programming:

- a function is a type
- on which various operations are defined
- which can be used to solve optimization problems

so use Julia for your mathematical programming:

- Julia has a tremendous ecosystem of optimization software
- use it!

more information (and code!)

- JuliaOpt: `http://www.juliaopt.org/`
- Convex: `http://www.github.com/JuliaOpt/Convex.jl`
- LowRankModels: `http://www.github.com/madeleineudell/LowRankModels.jl`

more on Convex and LowRankModels at JuliaCon:

- 3:52pm today: Mihir Paradkar on GraphGLRMs
- (yesterday) 5:09pm: Ayush Pandey on complex numbers in Convex

Data table

age	gender	state	diabetes	education	⋯
22	F	CT	?	college	⋯
57	?	NY	severe	high school	⋯
?	M	CA	moderate	masters	⋯
41	F	NV	none	?	⋯
⋮	⋮	⋮	⋮	⋮	

- detect demographic groups?
- find typical responses?
- identify related features?
- impute missing entries?

LowRankModels: behind the scenes

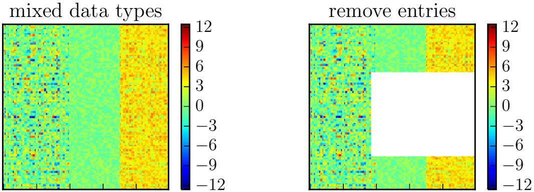
LowRankModels is a framework for specifying and solving matrix factorization problems

LowRankModels handles problems with a fixed, but complex, structure: matrix factorization problems with missing data

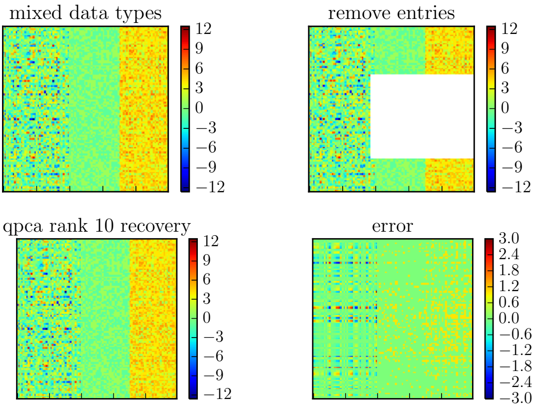
what properties of functions does LowRankModels use?

- evaluate
- gradient
- prox

Impute heterogeneous data



Impute heterogeneous data



Impute heterogeneous data