What is an optimization problem?

the optimization contract:

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

![Graph of a function with a minimum point]
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 of $f$?

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

- set derivative to zero \dots if $\frac{df}{dx}$ has an analytical formula
- gradient descent / backprop \dots 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 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 of $f$?

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

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

\[
f(x) = x^2
\]

\[
\begin{array}{c|c}
\text{f(1)} & 1 \\
\text{f(2)} & 4 \\
\end{array}
\]

- 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

minimize $f_0(x)$
subject to $f_i(x) \leq 0, \quad i = 1, \ldots, m_1$
$h_i(x) = 0, \quad i = 1, \ldots, m_2$
$x \in C$

- objective $f_0$
- inequality constraints $f_i$
- equality constraints $h_i$
- domain $C$

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., GLPK, 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)

▶ \( x + y \quad \Rightarrow \quad (+, (x, y)) \)
▶ \( x[1] + x[2] \quad \Rightarrow \quad (+, ((\text{index}, (x, 1)), (\text{index}, (x, 2)))) \)
▶ \( \log(x + 7y) \quad \Rightarrow \quad (\log, (+, (x, (*, (7, y))))) \)

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

Problem

Objective

minimize

CvxExpr

norm_inf

x

Array{CvxConstr}

CvxConstr

CvxExpr

==

5

CvxConstr

<=

x[3]

x[2]

x[1]

x[2]
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

implies sublevel sets

$\{x : f(x) \leq \alpha\}$ are convex sets
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)*
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 (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)
A function \textit{vexity} is defined on each data type (variable, constant, functions, constraints, problems) to return its \textit{vexity}: constant, affine, convex, concave, or not DCP.

\textbf{base case:}

- \textit{Constant}. Constants are \textbf{constant}.
- \textit{Variable}. Variables are \textbf{affine}.
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.
Composition rules are implemented as **arithmetic on vexities**: 

\[
\text{convex function} \quad \text{nondecreasing} \quad \text{in} \quad \text{convex expression} \quad \text{is} \quad \text{convex}
\]

\[
\text{ConvexVexity} + \quad \text{NonDecreasing} \quad * \quad \text{ConvexVexity} \quad == \quad \text{ConvexVexity}
\]
DCP in action

demo:

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

Convex transforms optimization problems to conic form:

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad b - Ax \in K,
\end{align*}
\]

where \( K \) is a **convex cone**:

\[ x \in K \iff rx \in K \text{ for any } r > 0. \]

**examples:**

- zero cone \( K = \{0\} \)
- positive orthant \( K = \{x : x_i \geq 0, \ i = 1, \ldots, n\} \)
- second order cone \( K = \{(x, t) : \|x\|_2 \leq t\} \)
- positive semidefinite (PSD) cone 
  \( K = \{X : X = X^T, \ v^T X v \geq 0, \ \forall v \in \mathbb{R}^n\} \)
- products of cones
Why conic form?

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad b - Ax \in K,
\end{align*}
\]

advantages:

- efficiently grok the structure of problem
- fast solvers
Conic form for expressions

epigraph conic form for expressions:

\[ f(x) = \min_{C[x, t]} \quad \text{with variable } t \]
\[ \text{subject to } A[x, t] \in \mathcal{K} \]

(note: “objective” can be vector valued)

- function can be represented by tuple

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

**base case:**

- **Constant.**
  
  - $\text{min } 3$
  
  $3 = \text{ with variable } \emptyset$
  
  $\text{subject to } \emptyset$

- **Variable.**
  
  - $\text{min } x$
  
  $x = \text{ with variable } \emptyset$
  
  $\text{subject to } \emptyset$
Conic form: inductive rule

**inductive rule:** if

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

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

then

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

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

**inductive rule:** if

\[
\begin{align*}
\min & \quad C^f[y, t^f] \\
& \text{with variable } t^f \\
& \text{subject to } A^f[y, t^f] \in \mathcal{K}^f, \\
\min & \quad C^g[x, t^g] \\
& \text{with variable } t^g \\
& \text{subject to } A^g[x, t^g] \in \mathcal{K}^g
\end{align*}
\]

then

\[
\begin{align*}
\min & \quad C^f[C^g I][x, t^g, t^f] \\
& \text{with variable } t^g, t^f \\
& \text{subject to } A^f[C^g I][x, t^g, t^f] \in \mathcal{K}^f \\
& A^g[x, t^g] \in \mathcal{K}^g
\end{align*}
\]

**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:

\[
f(g(x)) = \min_{t^g, t^f} \quad C^f [C^g I][x, t^g, t^f] \]

subject to

\[
A^f [C^g I][x, t^g, t^f] \in \mathcal{K}^f \]
\[
A^g [x, t^g] \in \mathcal{K}^g
\]

in code:

```python
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/\ntypes-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/

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

<table>
<thead>
<tr>
<th>age</th>
<th>gender</th>
<th>state</th>
<th>diabetes</th>
<th>education</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>F</td>
<td>CT</td>
<td>?</td>
<td>college</td>
</tr>
<tr>
<td>57</td>
<td>?</td>
<td>NY</td>
<td>severe</td>
<td>high school</td>
</tr>
<tr>
<td>?</td>
<td>M</td>
<td>CA</td>
<td>moderate</td>
<td>masters</td>
</tr>
<tr>
<td>41</td>
<td>F</td>
<td>NV</td>
<td>none</td>
<td>?</td>
</tr>
</tbody>
</table>

- 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

mixed data types

remove entries
Impute heterogeneous data

mixed data types

remove entries

qpca rank 10 recovery

glrm rank 10 recovery

error
Impute heterogeneous data

mixed data types

remove entries

qpca rank 10 recovery

glrm rank 10 recovery

error