Optimization

Hyun Min Kang
July 16th, 2019
Michigan Big Data Summer Institute
What is optimization?

• From Merriam-Webster:
  • (noun) an act, process, or methodology of making something (such as a design, system, or decision) as fully perfect, functional, or effective as possible.
  • specifically : the mathematical procedures (such as finding the maximum of a function) involved in this

• A mathematical definition:
  • Given $f: A \to R$,
    find $x^* \in A$
    such that $f(x)$ is minimized at $x = x^*$

(Note: maximization can be easily flipped to a minimization)
Numerical optimization: examples with closed-form solutions

• Unconstrained optimization
  • \( f(x) = x^2 + 2x + 2 \)
  • \( f(x) = (x + 1)^2+1 \geq 1 \) (equality holds at \( x = -1 \))

• Constrained optimization
  • \( f(x) = x^2 + 2x + 2, \quad (x \geq 0) \)
  • \( f'(x) = 2x + 2 > 0 \) when \( x \geq 0 \), so monotonically increasing.
  • \( f(x) \) is minimized at \( x = 0 \), and \( f(x) \geq 2 \).

You are very lucky if your real-world optimization problem has a closed-form solution
Combinatorial optimization: an example

Manhattan Tourist Problem:
Find a path with the minimum total cost.

Today, we will not cover combinatorial optimization
Mathematical optimization problem

• Minimize the objective function
  \[ f_0(x) \]

• subject to the constraints
  \[ f_i(x) \leq b_i, \quad i \in \{1,2,\ldots,m\} \]

• where
  • optimization variable \( x = (x_1, \ldots, x_n) \)
  • objective function \( f_0: \mathbb{R}^n \to \mathbb{R} \)
  • constraint function \( f_i: \mathbb{R}^n \to \mathbb{R}, \quad i \in \{1,\ldots,m\} \)
Why study optimization?

• It is important to **formulate what you want** as an optimization problem.
  
  (e.g. LASSO) Given $X \in \mathbb{R}^{n \times p}$, $y \in \mathbb{R}^n$, and $\lambda \geq 0$, we want to find $\beta \in \mathbb{R}^p$ that minimizes

  $$f(\beta) = \|y - X\beta\|_2 + \lambda \|\beta\|_1 = (y - X\beta)^T(y - X\beta) + \lambda \sum_{i=1}^p |\beta_i|$$

• It is even more important **find out how to solve** the optimization problem.
  
  • This may not be a fun activity for everybody, but useful for most of us.
Types of optimization problems

- By **type of solutions**
  - Numerical optimization
  - Combinatorial optimization

- By **number of variables**
  - Single-dimensional optimization
  - Multi-dimensional optimization

- By **randomness in algorithm**
  - Deterministic optimization
  - Stochastic optimization

- By **type of objective function**
  - Convex optimization
  - Non-convex optimization

- By **constraints**
  - Constrained optimization
  - Unconstrained optimization

- By **optimality of the solution**
  - Local optimization
  - Global optimization
Optimization: **three** key questions

1. How can I **formulate the problem** into an optimization problem?
   • Articulate your problem in mathematical terms.
   • In some cases, you may not even have realized that it is an optimization problem.

2. Do I know **how to obtain a solution** for the optimization problem?
   • Having an objective function does not automatically solve the problem.
   • Certain optimization problems are much harder than others.

3. Do I know what the **time complexity** of the method I chose is?
   • If you have big data, time complexity is one of the key factor to consider.
   • The solution should be not only possible but also feasible to obtain.
Example: maximum likelihood estimation (MLE)

• Likelihood function
  • $x$ : observed data
  • $\theta$ : model parameter
  • $f (X = x; \theta)$ : probability density (mass) function
  • $L(\theta; x) = f (x; \theta)$ : likelihood function

• Maximum likelihood estimation (MLE) : find
  $$\hat{\theta} = \arg \max_\theta L(\theta; x)$$

• MLE is a useful across many areas of statistical inference.

• MLE is an instance of mathematical optimization problems
Example: MLE in logistic regression

• Given
  • $y \in \{-1,1\}^n$
  • $X \in \mathbb{R}^{n \times p}$
  • $\beta \in \mathbb{R}^p$

• Likelihood function
  \[
  L(\beta) = \prod_{i=1}^{n} \Pr(y_i = 1; X, \beta) = \prod_{i=1}^{n} \frac{1}{1 + \exp(-y_i x_i^T \beta)}
  \]

• Log-likelihood function
  \[
  l(\beta) = \log L(\beta) = \sum_{i=1}^{n} \log \left( \frac{1}{1 + \exp(-y_i x_i^T \beta)} \right)
  \]

This is an unconstrained optimization problem
A simple 1-dimensional case

• Suppose that $p = 1$, then the log-likelihood function becomes

$$l(\beta) = \log L(\beta) = \sum_{i=1}^{n} \log \left[ \frac{1}{1 + \exp(-y_i x_i \beta)} \right]$$

• Maximum likelihood estimate (MLE) is

$$\hat{\beta} = \arg \max_{\beta} \left( \sum_{i=1}^{n} \log \left[ \frac{1}{1 + \exp(-y_i x_i \beta)} \right] \right)$$

Q. Does a close-form solution exist?
Example: 1-d MLE in logistic regression

```r
n <- 1000  # make 1,000 arbitrary example points
x <- rnorm(n)  # where x is normally distributed
y <- rbinom(n, 1, 1/(1+exp(-x))) * 2 - 1  # y follows univariate logistic model of x
df <- data.frame(x=x, y=y)
library(ggplot2)
ggplot(df, aes(x=x, y=y)) + geom_point(position=position_jitter(w=0, h=0.05), alpha=0.3)
```

See the examples in the R markdown file in Canvas
Example data (jittered)
Likelihood function

\[ l(\beta) = \log L(\beta) = \sum_{i=1}^{n} \log \left( \frac{1}{1 + \exp(-y_i x_i \beta)} \right) \]

```r
llk1 <- function(beta, x, y) {
  return(-sum(log(1+exp(0-y*x*beta))))
}
betas <- (-100:100)/25
df <- data.frame(beta=betas, llk=sapply(betas, function(b) { llk1(b, x, y) }))
ggplot(df, aes(beta, llk)) + geom_line()
```
Visualization of the likelihood function
Single-dimensional optimization problem

• Given
  • \( f(x) \) : the objective function
  • We do not know how the function is shaped \textit{a priori}.
  • Evaluation of \( f(x) \) could be expensive – needs as few evaluations as possible.

• Want
  • Find \( x \) that minimizes \( f(x) \)
  • How difficult can this be?
Single-dimensional optimization could be tricky

Local vs. Global optimization

• **Globally optimal point**: \( x \) is globally optimal if
  \[
  f_0(x) = \inf_{z \in \mathcal{X}} f_0(z)
  \]
  where \( \mathcal{X} \) is a set of values that satisfy the constraints

• **Locally optimal point**: \( x \) is locally optimal if there exists \( R > 0 \) such that
  \[
  f_0(x) = \inf_{|z-x| \leq R, z \in \mathcal{X}} f_0(z)
  \]
Bracketing: from global to local optimization

• The goal of bracketing is to find three points such that

\[ a < b < c \]
\[ f(b) < f(a) \]
\[ f(b) < f(c) \]

• Once such \(a, b, c\) are identified for a continuous function \(f(\cdot)\), there should be at least one locally optimal point between \(a\) and \(c\).
Golden section search
Minimizing the worst-case damage

The next interval will have length either $1 - w$ or $w + z$.

Optimal condition must satisfy the following two conditions:

1. $1 - w = w + z$
2. $\frac{z}{1 - w} = w$

Solving the equations will lead to the golden ratio $w = \frac{3 - \sqrt{5}}{2} = 0.38197$.

This will guarantee that the interval size will reduce by $\sim 38\%$ at each step.
**Algorithms for single-dimensional optimization**

- **Golden section search**
  - At each iteration, the bracket size reduces by ~38%.
  - Guaranteed convergence, but could be slow.

- **Parabola method**
  - Approximate a quadratic function based on 3 points.
  - Often faster than golden search, but may not converge.

- **Brent’s method**
  - Combination of parabola method and golden search.
  - Most widely used method for single-dimensional optimization.
Finding MLE using **Brent’s** method

```r
## make sure to flip the sign of the objective function
## to convert the problem into minimization problem.
print(optimize(function(b) { 0-llk1(b, x, y)}, interval=c(-4,4)))

## $minimum
## [1] 0.9865495
##
## $objective
## [1] 601.4521
```
Multi-dimensional optimization

• More **common** type of optimization problems
  • Many variables need to be estimated together.

• A LOT **harder** than single-dimensional optimization
  • The search space is A LOT larger, especially for high-dimensions
  • Checking for local minimum is more complicated.
  • Checking for global minimum is even more complicated.

• A LOT more **diversity** in the available algorithms.
Ways to optimize *multi-dimensional* function

• If only the *objective* function is available (no gradient or Hessian)
  • Nelder-Mead algorithm

• If the *objective* function and *gradient* are available
  • Gradient descent (coordinate, batch, stochastic) algorithms
  • Quasi-Newton methods : BFGS or L-BFGS-B algorithms

• If *objective* function, *gradient*, and *Hessian* are available
  • Newton’s method
    
    *but Hessian is expensive to compute for high-dimensions, so not very common.*

*These are generic methods, and many other context-specific methods are available*
Nelder-Mead algorithm

• A general-purpose multi-dimensional minimization method.
• Published in 1965, and cited >29,000 times to date.
• Simple to use - does not require derivatives.
• Works quite well in practice for low (e.g. several) dimensions.
• No theoretical guarantee for convergence (either local or global)
• Typically slower than other methods that leverage gradient.
Basic operations of Nelder-Mead algorithm

Illustration of Nelder-Mead Algorithm

https://userpages.umbc.edu/~rostamia/2017-09-math625/images/nelder-mead.gif
Example of multi-dimensional optimization

• Multi-dimensional logistic regression

\[ l(\beta) = \log L(\beta) = \sum_{i=1}^{n} \log \left( \frac{1}{1 + \exp(-y_i x_i^T \beta)} \right) \]

• A straightforward extension of the 1-d logistic regression
Example R code

n <- 1000  # make 1,000 arbitrary example points
beta <- c(0.3, 0.1, 0.03, 0, 0)  # These are true effect sizes
p <- length(beta)  # p is the dimension of the variables
X <- matrix(rnorm(n*p), n, p)  # X is a (n x p) matrix of predictor variables
y <- rbinom(n, 1, 1/(1+exp(0-X%*%beta))) * 2 - 1  # y is a size n vector of -1/1
Example of simulated \textbf{data}

\begin{verbatim}
head(X)

## [1,]  0.4338128 -0.55187016  1.20380712 -1.302355079 -0.20147809
## [2,]  0.3658146  0.25108105  2.78898510  1.842643877  1.78737581
## [3,] -1.1087396  1.42582282 -0.09182399 -1.303832621  0.25927485
## [4,] -0.1451349 -0.06481368 -0.85645621 -0.939867038  0.95440592
## [5,]  0.3856186  0.25909865  0.88808601  0.755552557  0.65323539
## [6,] -0.6401423 -1.05290198 -1.59694764 -0.003498132  0.09643192

data(y)

## y
## -1  1
## 469 531
\end{verbatim}
**Likelihood function**

\[
llk2 \leftarrow \textbf{function}(b, X, y) \{
    \textbf{return} \left( -\text{sum}(\log(1+\exp(-y*(X\%\%b)))) \right)
\}
\]

\[
llk2(c(0,0,0,0,0), X, y)
\]

```r
## [1] -693.1472
```

Null likelihood

\[
llk2(c(0.3,0.1,0.03,0,0), X, y)
\]

```r
## [1] -677.0216
```

Likelihood at the true parameter

**Q. Is this the MLE?**
Nelder-Mead is implemented in `optim()`

General-purpose Optimization

Description

General-purpose optimization based on Nelder-Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization and simulated annealing.

Usage

```r
optim(par, fn, gr = NULL, ..., 
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", 
                 "Brent"), 
      lower = -Inf, upper = Inf, 
      control = list(), hessian = FALSE)

optimHess(par, fn, gr = NULL, ..., control = list())
```

Arguments

- **par**: Initial values for the parameters to be optimized over.
- **fn**: A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
- **gr**: A function to return the gradient for the "BFGS", "CG" and "L-BFGS-B" methods. If it is `NULL`, a finite-difference approximation will be used. For the "SANN" method it specifies a function to generate a new candidate point. If it is `NULL` a default Gaussian Markov kernel is used.
- **...**: Further arguments to be passed to `fn` and `gr`.
- **method**: The method to be used. See 'Details'. Can be abbreviated.
Logistic MLE using Nelder-Mead algorithm

```r
optim(c(0,0,0,0,0), function(b) { -llk2(b, X, y)})
```

```r
## $par
## [1] 0.35057819 0.09064599 -0.03585259 0.10049316 -0.06493131
##
## $value
## [1] 674.4148
##
## $counts
## function gradient
## 312 NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```
Optimization with gradients

• Gradient is a multivariate generalization of derivative

\[ \nabla f_0(x) = \left( \frac{\partial}{\partial x_1} f_0(x), \ldots, \frac{\partial}{\partial x_p} f_0(x) \right) \]

• For differentiable objective function, gradient is useful..
  • ... to approximate the “slope” of the objective function.
  • ... to reduce the number of function evaluations.
  • ... to achieve better convergence properties.
Coordinate descent algorithm

- Alternate each dimension for iterative update.
- Uses single-dimensional derivative to determine the direction and size of update.
- One of the simplest method.
- Does not work at all in some cases (where no improvement can be made using a single dimension).
Gradient descent algorithm

- Also called steepest descent algorithm.

- Parameters are updated to the direction proportional to the negative gradient of the objective function

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma^{(t)} \nabla f_0(\mathbf{x}^{(t)})$$

- Choosing the step size is one of the tricky part in implementation

https://en.wikipedia.org/wiki/Gradient_descent
Stochastic gradient descent (SGD)

• For very large data, calculating gradient across all data can be time-consuming.
• In many cases, the object-function can be separated into a summation form.

\[ f_0(x; D) = \sum_{i=1}^{m} f_0^{(i)}(x; d_i) \]

• Then, the gradient can also be represented into a summation form.

\[ \nabla f_0(x; D) = \sum_{i=1}^{m} \nabla f_0^{(i)}(x; d_i) \]

• Stochastic gradient descent compute gradient from partial data (single observation or mini-batch) to expedite the speed of update at the expense of smaller improvement at each update.
Types of gradient descent algorithms

• **Batch** gradient descent
  • Use all observations to compute gradient $\nabla f_0(x^{(t)}; D)$
  • Takes longer to compute, but gives a right direction to update parameters.

• **Stochastic** gradient descent
  • Update the parameters using a single-sample gradient $\nabla f_0^{(i)}(x^{(t,i)}; d_i)$
  • Gradient can be computed faster, but update can go in a wrong direction.

• **Mini-batch** gradient descent
  • Compute gradient using a small batch of samples.
  • Gradient is more informative than using only a single sample, at the expense of increased cost of computation.
Stochastic gradient descent - Illustration

Image by Imad Dabbura from towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3
Benefits of stochastic gradient descent

- SGD typically converges much faster than batch update algorithms per accessed data points.

- SGD converges fast at the beginning, but may converge slowly at the end.

Stochastic gradient descent for **logistic regression**

Example from R. Tibsharani’s lecture
Quasi-Newton methods

• Gradient descent – uses gradient to determine the next point
  \[ \mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma^{(t)} \nabla f_0(\mathbf{x}^{(t)}) \]

• Newton’s method – uses (expensive) 2\textsuperscript{nd}-order information.
  \[ \mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \left[ \nabla^2 f_0(\mathbf{x}^{(t)}) \right]^{-1} \nabla f_0(\mathbf{x}^{(t)}) \]

• Quasi-Newton methods approximate Hessian using gradients
  \[ \mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma^{(t)} \left[ H^{(t)} \right]^{-1} \nabla f_0(\mathbf{x}^{(t)}) \]

where H are iteratively updated using previous gradients
Broygen-Fletcher-Goldfarb-Shanno (BFGS) update.

- **BFGS algorithm**
  - Let $s = x^{(t)} - x^{(t-1)}$ and $y = \nabla f_0 (x^{(t)}) - \nabla f_0 (x^{(t-1)})$
  - The BFGS update approximate Hessian using the following rule

$$
H^{(t)} = H^{(t-1)} + \frac{yy^T}{y^Ts} - \frac{H^{(t-1)}ss^TH^{(t-1)}}{s^T H^{(t-1)} s}
$$

- **L-BFGS-B algorithm**
  - Extended version of BFGS with two additional features:
    - **Limited memory** – compute H more rapidly with less memory.
    - **Box constrains** – Allow box-like constraints in the optimization problem.
BFGS and L-BFGS-B are implemented in `optim()`

Optim {stats}

R Documentation

General-purpose Optimization

Description

General-purpose optimization based on Nelder–Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization and simulated annealing.

Usage

```r
optim(par, fn, gr = NULL, ...,
        method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN",
                   "Brent"),
        lower = -Inf, upper = Inf,
        control = list(), hessian = FALSE)
```

```r
optimHess(par, fn, gr = NULL, ..., control = list())
```

Arguments

`par`
Initial values for the parameters to be optimized over.

`fn`
A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.

`gr`
A function to return the gradient for the "BFGS", "CG" and "L-BFGS-B" methods. If it is NULL, a finite-difference approximation will be used.
Gradient of logistic objective function

\[ f_0(\beta) = -l(\beta) = \sum_{i=1}^{n} \log[1 + \exp(-y_i x_i^T \beta)] \]

\[ \nabla f_0(\beta) = \sum_{i=1}^{n} \frac{-y_i x_i^T \exp(-y_i x_i^T \beta)}{1 + \exp(-y_i x_i^T \beta)} \]

```
logistic.gradient <- function(b, X, y) {
  tmp <- exp(-y*(X%*%b))
  return( colSums(matrix( -y*tmp/(1+tmp), nrow(X), ncol(X) ) * X) )
}
```
Running **L-BFGS-B** Algorithm

```r
optim(c(0,0,0,0,0),
      fn = function(b) { 0-llk2(b, X, y)},
      gr = function(b) { logistic.gradient(b, X, y)},
      method="L-BFGS-B")
```

```
## $par
## [1] 0.35075554 0.09069752 -0.03581004 0.10039737 -0.06488337
##
## $value
## [1] 674.4148
##
## $counts
## function gradient
##     6     6
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```
So far we have learned...

• **Single-dimensional** optimization  
  • Golden section search  
  • Brent’s method

• **Multi-dimensional** optimization  
  • Nelder-Mead algorithm  
  • Coordinate gradient descent  
  • Batch (steepest) gradient descent  
  • Stochastic gradient descent  
  • Quasi-Newton methods: BFGS, L-BFGS-B

*These are “generic” algorithms that do not depend on properties of the objective function*
Specialized optimization methods

• There are many optimization methods that are specialized for particular subset of optimization problems.

• These methods exploit the intrinsic structure of the problems to more accurately and/or efficiently solve the optimization problems.

• Some specialized optimization methods are still quite general (i.e. applicable to a wide range of similar problems), while some others are tailored only to a particular instance of problem.
Some **examples** of specialized optimization

• For logistic regression, the standard optimization used is “Iteratively Reweighted Least Squares” (IRWS)

• For LASSO, where we optimizes the following function

$$f(\beta) = \|y - X\beta\|_2 + \lambda \|\beta\|_1 = (y - X\beta)^T (y - X\beta) + \lambda \sum_{i=1}^{n} |\beta_i|$$

the “least-angle regression” (LARS) is the algorithm used often.

*We won’t have time to look into the details of these methods, but there are reasons why these algorithms are well-suited for these particular problems.*
Some widely used optimization methods

• Expectation-Maximization (E-M) algorithm
• Simulated annealing
• Linear programming
• Quadratic programming
• Semidefinite programming
• Alternating direction method of multipliers (ADMM)
E-M Algorithm: Overview

- Iterative algorithm for solving MLE problems with missing data

- E-M algorithm is particularly useful when:
  - There are missing (unobserved) data
  - The MLE is analytically intractable if missing data is unobserved
  - The MLE would analytically be tractable if missing data was observed.

- A popular and highly cited (>55,000 times) method.
The **basic** E-M strategy

• Types of data
  • **Complete** data \((x, z)\) : what we would like to have
  • **Observed** data \(x\) : individual observations
  • **Missing** data \(z\) : hidden/missing values

• The E-M algorithm overview
  1. **Initialize** the parameter \(\theta^{(t)}\)
  2. **E-step** : calculate the distribution of hidden value using current parameter \(\theta^{(t)}\)
  3. **M-step** : update the parameter \(\theta^{(t+1)}\) to maximize the expected log-likelihood.
  4. **Repeat** step 2-3 until convergence.
Th Expectation-Maximization algorithm

• E-step

Given $\theta^{(t)}$ and $x$, calculate the following quantity:

$$w(z|x, \theta^{(t)}) = \frac{L(\theta^{(t)}|x, z)}{L(\theta^{(t)}|x)} = \frac{f(x, z|\theta^{(t)})}{g(x|\theta^{(t)})}$$

• M-step

Find $\theta^{(t+1)} = \arg \max_\theta Q(\theta|\theta^{(t)})$ that maximizes the expected log-likelihood

$$Q(\theta|\theta^{(t)}) = \mathbb{E}_Z \left[ \log L(\theta|x, Z)|\theta^{(t)}, x \right] = \int_Z w(z|x, \theta^{(t)}) \log L(\theta|x, z)dz$$
Key property of the E-M algorithm

- The expected log-likelihood function satisfies that
  \[ g_t(\theta) \leq \log L(\theta|x) \text{ and } g_t(\theta^{(t)}) = \log L(\theta^{(t)}|x) \]

- The M-step maximizes the surrogate function, making the likelihood always increase at each iteration.

\[ \theta^{(t+1)} = \arg \max_{\theta} g^{(t)}(\theta) \]

\[ L(\theta^{(t+1)}|x) \geq L(\theta^{(t)}|x) \]

Example – Gaussian mixture model
Gaussian mixture model with true labels
Labels inferred from the E-M algorithm
Two dimensional Gaussian mixture
2D Gaussian mixture with true labels
Inferred labels with E-M algorithm
Challenges in **hill-climbing** methods

Overcoming the challenge: chaotic jump

Annealing

• Annealing is a manner in which crystals are formed.
• Gradual cooling of liquid can form crystal lattice

Simulated annealing

• Concept
  • Numerical optimization procedure which aims for \textit{global} optimization.
  • Use \textit{analogy} of thermodynamics

• Key idea
  • Incorporates \textit{temperature} parameter into the optimization procedure
  • At high temperature, \textit{explore} the parameter space
  • At low temperature, \textit{restrict} exploration.
Updates in simulated annealing

• Given a temperature, assume a probability proportional to Boltzmann factor

\[ P(\theta) \propto \exp \left( -\frac{f_0(\theta)}{T} \right) \]

• When updating parameters from \( \theta_0 \) to \( \theta_1 \), accept the change probabilistically

\[
\min \left( 1, \frac{P(\theta_1)}{P(\theta_0)} \right) = \min \left[ 1, \exp \left( -\frac{f_0(\theta_1) - f_0(\theta_0)}{T} \right) \right]
\]

• New parameter must be chosen based on a random procedure.
• If the solution was improved, always accept the new parameter.
• Otherwise, if T is high, the new parameter will be accepted with relatively often.
• When T is low, the new parameter will be very rarely accepted.
Illustration of simulated annealing procedure
Simulated annealing: highlights

• It is a global optimization method
  • Overcomes disadvantages in hill-climbing approach
  • Useful to avoid being trapped at local optima for high-dimensional problems

• It is a Markov-chain Monte-Carlo (MCMC) method
  • Randomly updates the parameter.
  • Probabilistically accept the new parameter based on Metropolis-Hastings (MH) procedure.

• Useful in solving a variety of optimization problems
  • Including combinatorial optimization such as the Traveling Salesman Problem
  • Implemented in optim() function in R
Convex optimization

- Convex optimization is a subset of mathematical optimization problem.
- Often there is much easier solution than non-convex optimization problems.
Example: Diet Problem

Doctor’s recommendation on diet restriction
• No more than 13,800mg of fat consumption
• At least 600mg, 300mg, 500mg of vitamin X, Y, Z consumptions.

Goal is to come up with most cost-effective diet plan

<table>
<thead>
<tr>
<th></th>
<th>Cost/unit</th>
<th>Fat/mg/unit</th>
<th>Vitamin X/mg/unit</th>
<th>Vitamin Y/mg/unit</th>
<th>Vitamin Z/mg/unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Food A</td>
<td>$5.00</td>
<td>800</td>
<td>50</td>
<td>10</td>
<td>150</td>
</tr>
<tr>
<td>Food B</td>
<td>$1.00</td>
<td>6,000</td>
<td>3</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>Food C</td>
<td>$6.00</td>
<td>1,000</td>
<td>150</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>Food D</td>
<td>$3.00</td>
<td>400</td>
<td>100</td>
<td>100</td>
<td>5</td>
</tr>
</tbody>
</table>
Formulating the problem \textit{mathematically}

- **Objective function**
  \[ f_0(x) = c^T x, \quad c = [5 \ 1 \ 6 \ 3]^T \]

- **Constraint functions**
  \[ f_1(x) = a_1^T x \leq b_1 \]
  \[ f_2(x) = a_2^T x \geq b_2 \]
  \[ f_3(x) = a_3^T x \geq b_3 \]
  \[ f_4(x) = a_4^T x \geq b_4 \]

\[
\begin{align*}
  a_1 &= [800 \ 6000 \ 1000 \ 400]^T, & a_2 &= [50 \ 3 \ 150 \ 100]^T \\
  a_3 &= [10 \ 10 \ 75 \ 100]^T, & a_4 &= [150 \ 35 \ 75 \ 5]^T \\
  b_1 &= 13800, & b_2 &= 600, & b_3 &= 300, & b_4 &= 550
\end{align*}
\]
Linear programming (LP)

- Optimization variable
  \[ \mathbf{x} = (x_1, \ldots, x_p) \in \mathbb{R}^p \]

- Objective function: minimize
  \[ f_0(\mathbf{x}) = c^T \mathbf{x} + d \]

- Constraint functions: subject to
  \[ \begin{align*}
  G\mathbf{x} & \leq \mathbf{h} \\
  A\mathbf{x} & = \mathbf{b} \\
  G & \in \mathbb{R}^{m \times p}, A \in \mathbb{R}^{q \times p}
  \end{align*} \]
Simplex algorithm for LP

• Optimal point occurs in one of the vertices of the simplex
• `boot::simplex()` in R can solve this problem efficiently
Quadratic programming (QP)

• Optimization variable

\[ \mathbf{x} = (x_1, \ldots, x_p) \in \mathbb{R}^p \]

• Objective function : minimize

\[ f_0(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{P} \mathbf{x} + \mathbf{q}^T \mathbf{x} + r \]

• Constraint functions : subject to

\[ \mathbf{G} \mathbf{x} \preceq \mathbf{h} \]
\[ \mathbf{A} \mathbf{x} = \mathbf{b} \]

\[ \mathbf{G} \in \mathbb{R}^{m \times p}, \mathbf{A} \in \mathbb{R}^{q \times p} \]
Optimally separating hyperplane
Maximizing the margin of hyperplane

- Minimize $\frac{1}{2} \|w\|^2 = \frac{1}{2} w^T w$
- Subject to $y_i (w^T x_i - b) \geq 1$
  for $i \in \{1, \ldots, n\}$

This is a quadratic programming (QP) problem
Support Vector Machine (SVM)

- To allow non-separable hyperplane, define a hinge loss
  \[ \xi_i = \max(0, 1 - y_i(w^T x_i - b)) \]

- Objective function for SVM
  \[
  \text{Minimize } \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \\
  \text{where } \xi_i = \max(0, 1 - y_i(w^T x_i - b))
  \]

- This can be represented as a QP, too
- Thus, SVM is a QP problem
Semidefinite programming (SDP)

- **Objective function**: minimize

\[ f_0(x) = c^T x \]

- **Constraint functions**: subject to

\[ F_0 + \sum_{i=1}^{p} x_i F_i \succeq 0 \]

\( \succeq 0 \) represents that the matrix is positive semidefinite (i.e. non-negative eigenvalues)
QP and SDP represent non-linear decision boundary

\[ F(x) \not\geq 0 \quad F(x) \geq 0 \]

Alternating Direction Method of Multipliers (ADMM)

• Consider convex functions $f$ and $g$ in the optimization problem.
  • Minimize $f(x) + g(z)$
  • Subject to $Ax + Bz = c$

• The problem assumes two sets of variables that are separable.
• The augmented Lagrangian is defined as
  $$L_\rho(x, z, \nu) = f(x) + g(z) + \nu(Ax + Bz - c) + \frac{\rho}{2} \|Ax + Bz - c\|_2^2$$
Iterative **update** steps for ADMM

\[ x^{k+1} \leftarrow \arg \min_x \mathcal{L}_\rho(x, z^k, \nu^k) \]

\[ z^{k+1} \leftarrow \arg \min_z \mathcal{L}_\rho(x^{k+1}, z, \nu^k) \]

dual update

\[ \nu^{k+1} \leftarrow \nu^k + \rho \left( Ax^{k+1} + B z^{k+1} - c \right) \]
Why ADMM?

• Because ADMM is VERY USEFUL!

• By separating objective function and constraints into two different functions, ADMM can be used to solve a wide variety of problems.

• Example problems solvable by ADMM
  • LASSO
  • Group LASSO
  • Linear programming
  • Quadratic programming
  • Non-negative matrix factorization (NMF)
  • and more…
Today: **Summary**

- **Generic optimization methods**
  - Golden section search
  - Brent’s methods
  - Nelder-Mead algorithm
  - Gradient descent algorithms
  - Quasi-Newton methods (BFGS, L-BFGS-B)

- **Specialized optimization methods**
  - E-M algorithm
  - Simulated annealing
  - Linear, Quadratic, and Semidefinite Programming
  - ADMM
Important things *not* covered today

- Markov-Chain Monte Carlo (MCMC) algorithm
- Metropolis-Hasting algorithm
- Gibbs sampler
- Lagrangian
- Lagrangian duality
- Karush-Kuhn-Tucker (KKT) condition
- Dual ascent
- RMSprop
- Adam
- Dynamic programming

*These are some keywords you may want to explore later on to learn more about optimization*
Optimization: *three* key questions

1. How can I *formulate the problem* into an optimization problem?
   - Articulate your problem in mathematical terms.
   - In some cases, you may not even have realized that it is an optimization problem.

2. Do I know *how to obtain a solution* for the optimization problem?
   - Having an objective function does not automatically solve the problem.
   - Certain optimization problems are much harder than others.

3. Do I know what the *time complexity* of the method I chose is?
   - If you have big data, time complexity is one of the key factors to consider.
   - The solution should be not only possible but also feasible to obtain.