Optimization Methods for Machine Learning

Haimonti Dutta
Department of Management Science and Systems, UB.
Email: haimonti@buffalo.edu

Acknowledgements

- **Slides:** Materials in the slides are obtained from several sources including:
  - Convex Optimization Algorithms. D. P. Bertsekas. Athena Scientific.
  - Optimization Methods for Large Scale Machine Learning. L. Bottou, F. E. Curtis and J. Nocedal.
  - Optimization for Machine Learning. S. Sra, S. Nowozin, and S. J. Wright.
  - The Interplay of Optimization and Machine Learning Research. K. P. Bennett and E. P. Hernández.

Acknowledgements

- **Discussions** Many people have advised and provided insightful comments including Hillol Kargupta, (late) David Waltz, Ashwin Srinivasan and many others.
- **Apologies**
  - If I do not get around to covering your favorite topic in optimization, or did not include it in the bibliography (drop me an email).
  - I will not get around to covering many topics including theory of convex optimization, non-convex optimization methods and much more.

Resources

The following things are available from a github repository:
https://github.com/Haimonti/optML

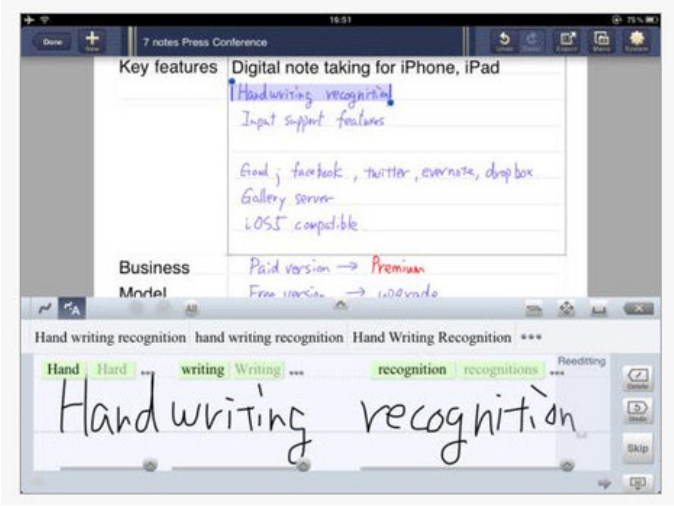
- Slides for this talk
- Sample R code

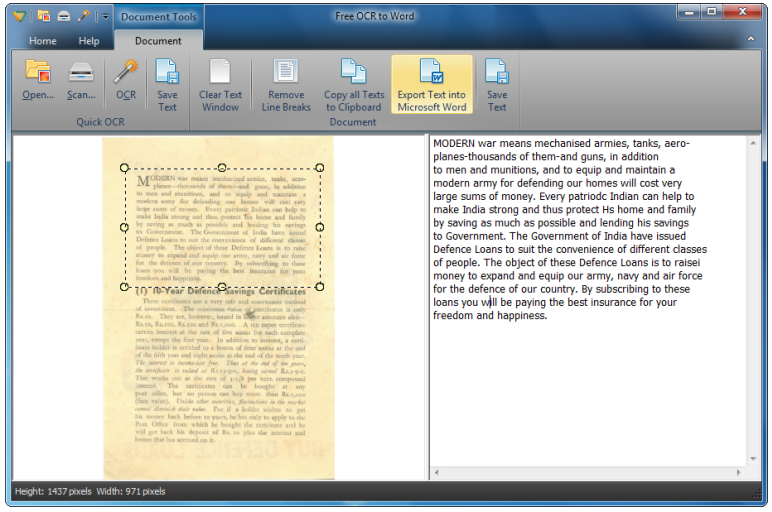
If you use the R-code for your work, please cite the above repository!
Drop me an email, so I can update the list of projects using the code.

Outline

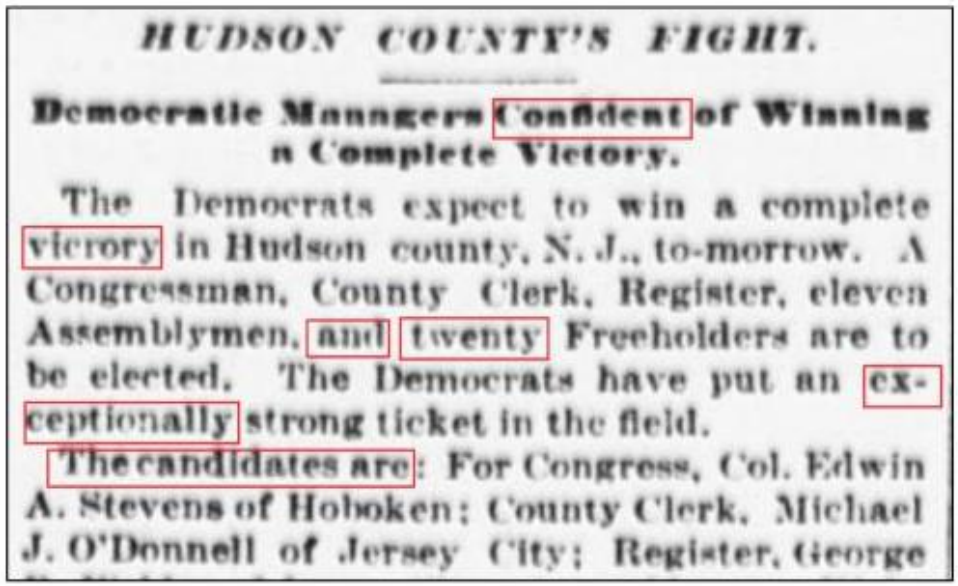
Case Study 1: Text Classification

- Optical Character Recognition (OCR) – electronic translation of handwritten, typewritten or printed text into machine translated images
- Wide applications in banking, healthcare, digital libraries, handwriting recognition,etc.[SBB12]
- Results in very large text databases





Example: Digital Humanities [DPL+11]



HUDSON COUNTY'S FIGHT.

Democratic Managers Confident of Winning a Complete Victory.

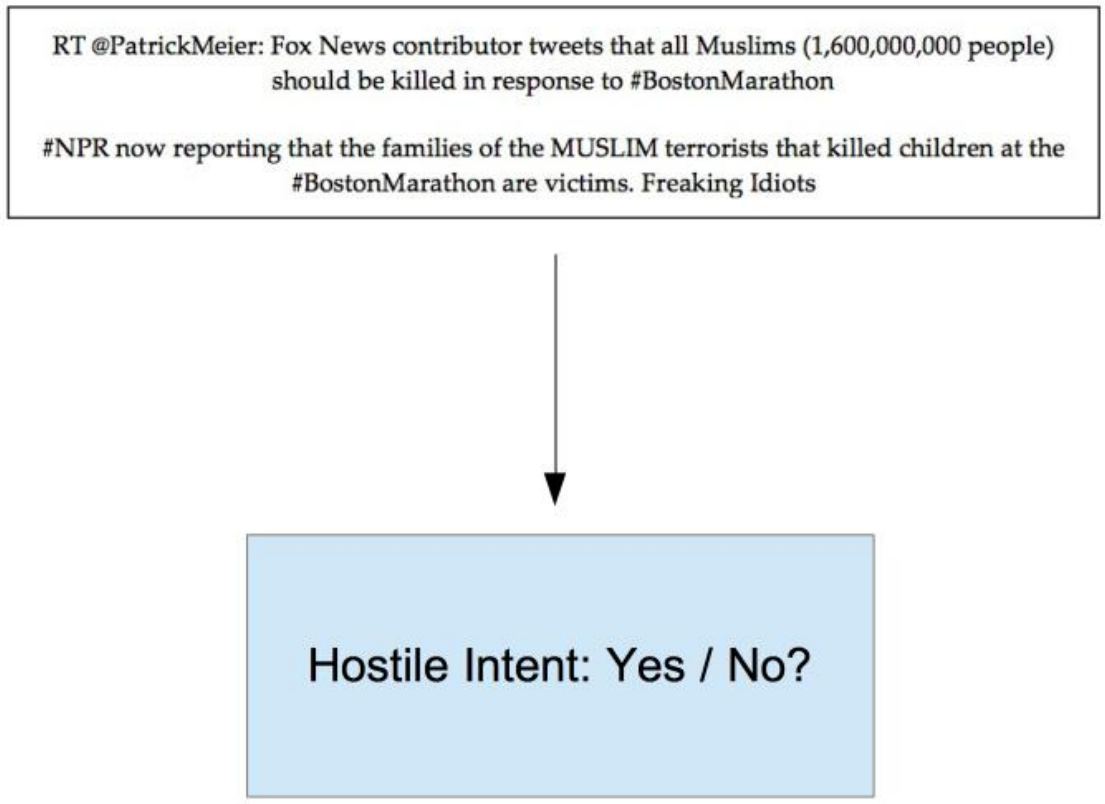
The Democrats expect to win a complete victory in Hudson county, N. J., to-morrow. A Congressman, County Clerk, Register, eleven Assemblymen, and twenty Freeholders are to be elected. The Democrats have put an exceptionally strong ticket in the field.

The candidates are: For Congress, Col. Edwin A. Stevens of Hoboken; County Clerk, Michael J. O'Donnell of Jersey City; Register, George

→ Politics?

- To which category does this news article belong?

Example: Analyzing Social Media Data



Text Classification and Convex Optimization

In statistical Machine Learning

- Collection of a sizable set of examples $\{(x_1, y_1), \cdots, (x_n, y_n)\}$
- x_i represents *features* of the text document (e.g. words it includes)
- $y_i = \pm 1$ is a scalar quantity, called *label* and indicates whether document belongs to a certain class (or topic of interest)
- **Goal:** Construct a prediction function h; measure its performance by counting how often the program prediction $h(x_i)$ differs from the correct prediction y_i

Search for a prediction function that *minimizes* the frequency of observed misclassifications

Empirical Risk Minimization

$R_n(h) = \frac{1}{n} \sum_{i=1}^{n} 1[h(x_i) \neq y_i]$ where

$$1(A) = \begin{cases} 1, & \text{if A = true,} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Rote Learning

Suppose we learnt a function that does the following:

$$h^{rote}(x) = \begin{cases} y_i, & \text{if } x = x_i \text{ for some } i \in \{1, \cdots, n\} \\ \pm 1, & \text{arbitrarily otherwise} \end{cases} \tag{2}$$

This clearly minimizes the function. But provides no performance guarantees on unseen examples!
We want a prediction function that *generalizes* the concepts learnt from examples.

Choice of prediction functions

- How can one choose amongst a carefully selected class of prediction functions (perhaps satisfying certain smoothness conditions and enjoying a convenient parametric representation)?
- Identify a small subset of viable candidates by minimizing R_n and attempting to choose an optimal h
- Impractical in large scale settings
- In practice – employ a continuous approximation through a *loss function* that measures a cost for predicting h when the true label is y

Text Classification - Revisited

- Usual Approach: Represent the text document using a *bag of words* representation
- Very large, sparse representation e.g. RCV1 uses a vocabulary $d = 47,152$ words to represent news stories which are less than 1000 words.

Text Classification - Revisited

- Consider linear discriminant functions
  - Prediction functions of the form $h(x; w, \tau) = w^T x - \tau$
  - Parameterized by $w \in R^d$ and $\tau \in R$
  - Measures of classifier performance: Compromise between Precision $P(y = 1|h(x) = 1)$ and Recall $P(h(x) = 1|y = 1)$
  - Measure accuracy by counting the number of times the sign function matches the correct label.
  - Recall, the sign function is defined as follows:

$$sgn(x) = \begin{cases} -1, & \text{if } x < 0, \\ 0, & \text{if } x = 0, \\ +1, & \text{if } x > 0. \end{cases} \tag{3}$$

  - Sgn(x) is a discontinuous function!
- Solution: Use continuous functions loss functions such as log loss. $log(h, y) = log(1 + exp(-hy))$. Add a regularization term to generalize.

Optimization problem formulation

$$min_{(w,\tau)\in R^d \times R} \ \frac{1}{n} \sum_{i=1}^{n} \underbrace{\ell(h(x_i; w, \tau), y_i)}_{\text{loss function}} + \underbrace{\frac{\lambda}{2} \parallel w \parallel_2^2}_{\text{regularization term}}$$

Categories of Optimization Methods

Optimization methods for machine learning fall into two categories:

- Batch
- Stochastic

Batch Methods

- Minimize R_n
- Start with an initial $w_1 \in R^d$
- Compute the gradient using the formula:
  $w_{k+1} = w_k - \alpha_k \nabla R_n(w_k) = w_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(w_k)$
- Also called *batch gradient* or *full gradient* method.

Outline

Why use iterative algorithms?

Iterative algorithms can be used for minimizing a function $f : R^d \rightarrow R$ over a set X. They can generate a sequence $\{x_k\}$ which will hopefully converge to an optimal solution.

Iterative Descent Algorithms

- Generate sequences $\{x_k\}$ according to $x_{k+1} = G_k(x_k)$, where $G_k(x_k) : R^d \to R^d$
- Algorithms depend on k and a starting point x_0
- G_k may depend on previous iterates $x_{k-1}, x_{k-2}, \cdots$
- Convergence of the algorithm (and the generated sequence) to a desired point is crucial.
- Study the rate of convergence of the algorithm - how fast do we reach the desired solution - i.e. how many iterations?

Stationary Iterative Algorithm

If G_k does not depend on k, we have $x_{k+1} = G_k(x_k)$
Find a solution of the equation $x = G(x)$.

An example - Gradient Iteration

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

- Assume there is a differential function $f : R^n \rightarrow R$.
- To find the unconstrained minima, optimality condition is $\nabla f(x) = 0$.
- α : Positive step size parameter. Ensures that iteration makes progress towards the solution set of the problem.

Let us find the minimum of a parabola

Problem 1: Find x that is minimum of $f(x) = 1.2(x-2)^2 + 3.2$ or, said another way, find $arg\ min_x f(x)$.

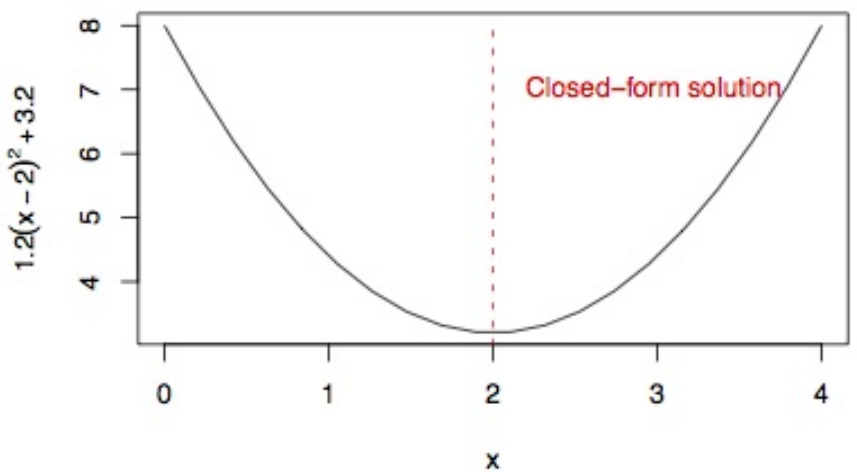
Solution: Take derivative, set it to 0 and solve for x!

$$f(x) = 1.2(x-2)^2 + 3.2$$

$$\frac{df(x)}{dx} = 1.2(2)(x-2) = 2.4(x-2)$$

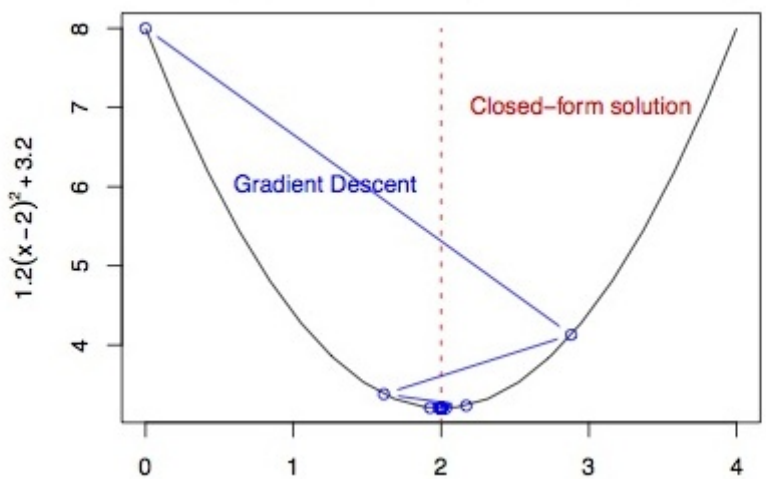
$$\frac{df(x)}{dx} = 0 = 2.4(x-2)$$

$$x = 2$$



More on gradient descent

- If we cannot find the derivative $\frac{df(x)}{dx}$ directly, how do we solve the problem?
- Start at some x value, use derivative at that value to tell which way to move, and repeat. Gradient descent.
- α is the factor of derivative to control how far to go.
- $\frac{df(x)}{dx} = 2.4(x-2)$
- Suppose, $x(0) = 0$
- In the k^{th} iteration, $x_{k+1} = x_k - \alpha 2.4(x-2)$

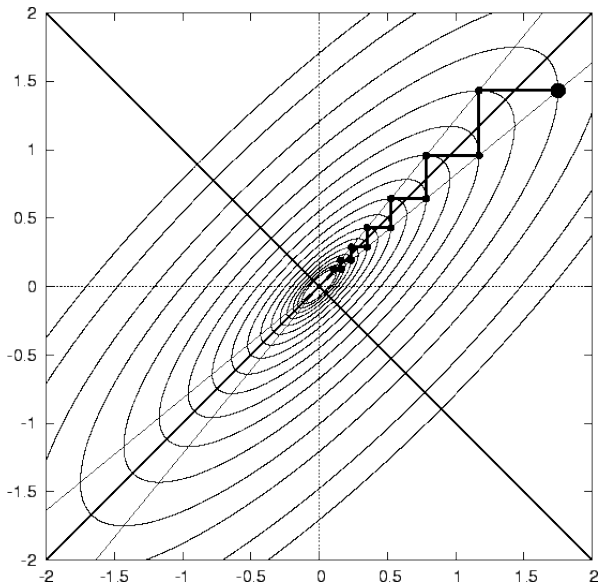


Two main ideas of iterative algorithms

- Iterative Descent
- Approximation

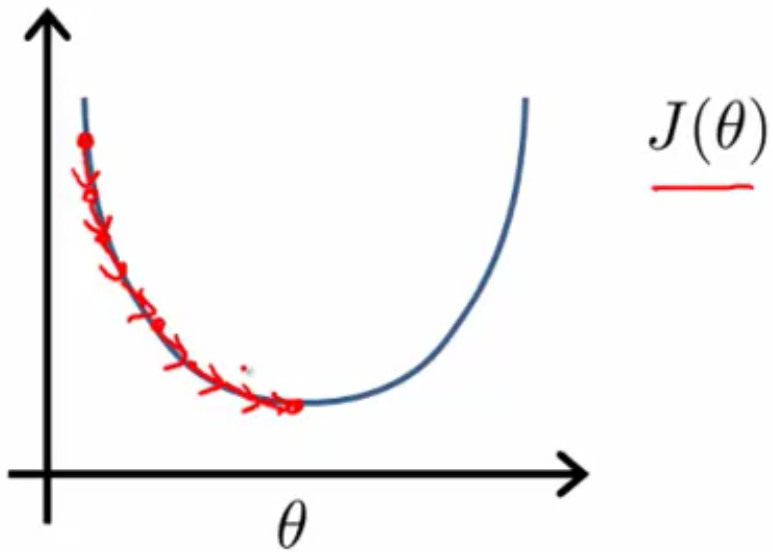
Iterative Descent

- The generated sequence $\{x_k\}$ is feasible i.e. $\{x_k\} \subset X$ and satisfies $\Phi(x_{k+1}) < \Phi(x_k)$, if and only if x_k is not optimal
- Φ is a merit function that measures the progress of the algorithm towards optimality.
- It is minimized only at optimal points i.e. arg $min_{x \in X} \Phi(x) = argmin_{x \in X} f(x)$
- Example: $\Phi(x) = f(x)$ and $\Phi(x) = inf_{x^* \in X^*} \parallel x - x^* \parallel$, X^* is the set of optimal points.



Approximation

- The generated sequence $\{x_k\}$ need not be feasible
- Obtained by solving at each k an approximation of the original minimization problem
- $x_{k+1} \in argmin_{x \in X_k} F_k(x)$ where $F_k(x)$ is a function that approximates f and X_k approximates X



Another Method: Newton's method

An iterative method for finding successively better approximations to the roots of a differentiable function.

Newton's method: The Algorithm

- Assume there is a function $f(x), x \in R$. The (first) derivative of $f(x)$ is $f'(x)$ and the second derivative is given by $f''(x)$.
- Let x_0 be an initial *guess* for the root of the function f.
- x^* is called the **stationary point** of $f(x)$.
- Taylor expansion
  $f(x_k) = f(x_k + \triangle x) \approx f(x_k) + f'(x_k)\triangle x + \frac{1}{2}f''(x_k)\triangle x^2$
- We want to find $\triangle x$ such that $(x_k + \triangle x)$ is a stationary point.
- Set the derivative of the Taylor expansion to zero i.e.
  $\frac{d}{d\triangle x}(f(x_k) + f'(x_k)\triangle x + \frac{1}{2}f''(x_k)\triangle x^2) =$
  $f'(x_k) + f''(x_k)\triangle x = 0$
- Choose $\triangle x = -\frac{f'(x_k)}{f''(x_k)}$

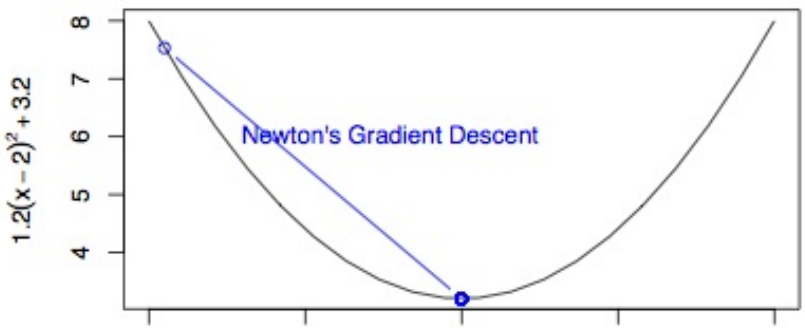
If we choose $x_{k+1} = x_k + \triangle x = x_k - \frac{f'(x_k)}{f''(x_k)}$ we will be closer to the stationary point.

- For a parabola, can get there much faster if we also know the second derivative, which is what?

$$\frac{df(x)}{dx} = f' = 2.4(x-2)$$
$$\frac{d^2f(x)}{dx^2} = f'' = 2.4$$

- and use Newton's method (see the Wikipedia entry for "Newton's method")

$$x(n) = x(n-1) - \frac{f'}{f''}$$
$$x(n) = x(n-1) - \frac{2.4(x-2)}{2.4}$$
$$x(n) = x(n-1) - (x-2)$$



More on Gradient Descent

- If the function is not a parabola, what can we do? Cannot solve directly for x. Can still do gradient descent. Can we always use Newton's method?
- No. The Hessian (second derivative) may be hard to compute.
- If not a parabola the second derivative information may lead you very far away. When?

Approximating the second gradient

- Say we have picked a direction, p, to go. Rather than compute the second derivative in that direction, we can approximate it using two first derivative values.

$$f''(x)p \approx \frac{f'(x+\alpha p) - f'(x)}{\alpha} \text{ for } 0 < \alpha << 1$$

- In practice, Moller found he had to modify this by adding λp where λ is set to a value for which the resulting approximated second derivative is well behaved.

$$f''(x)p \approx \frac{f'(x+\alpha p) - f'(x)}{\alpha} + \lambda p, \text{ for } 0 < \alpha << 1$$

- This gives us a way to scale the step size.

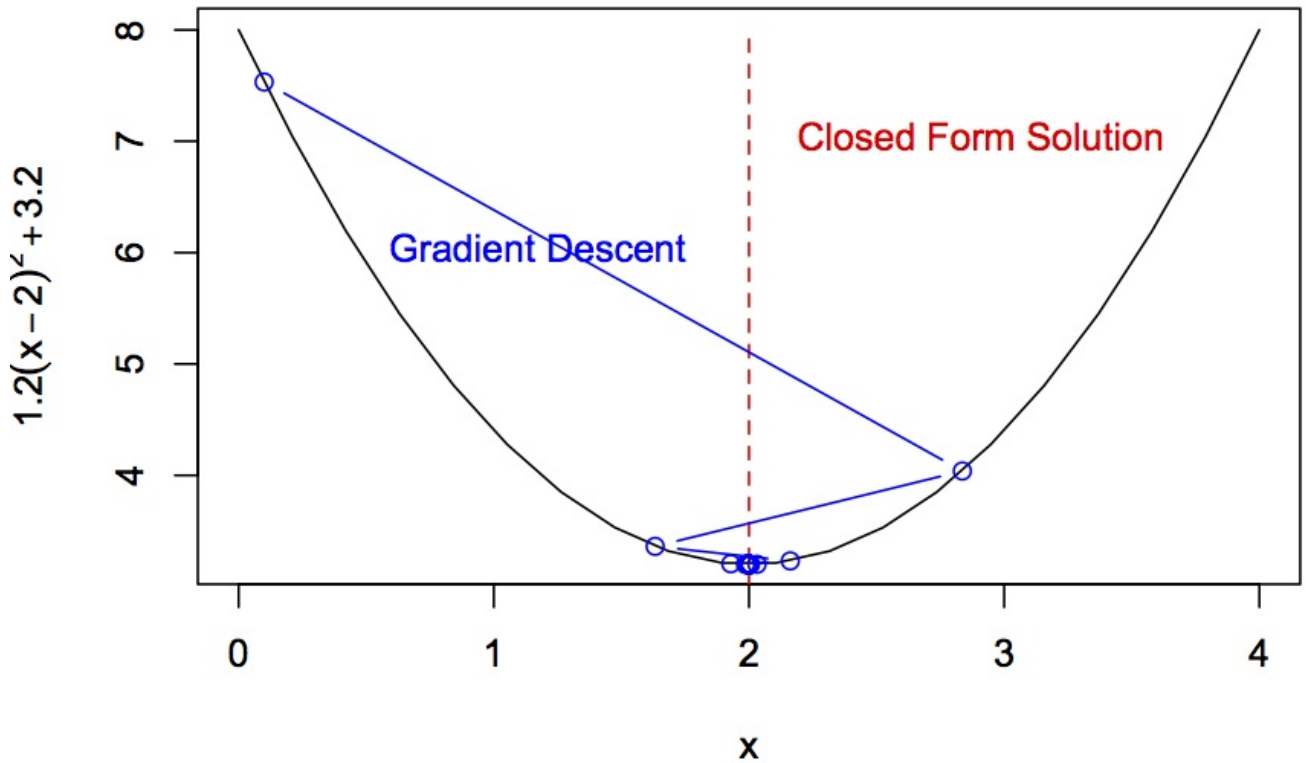
R Code - Gradient Descent

gradDescent

<sections>

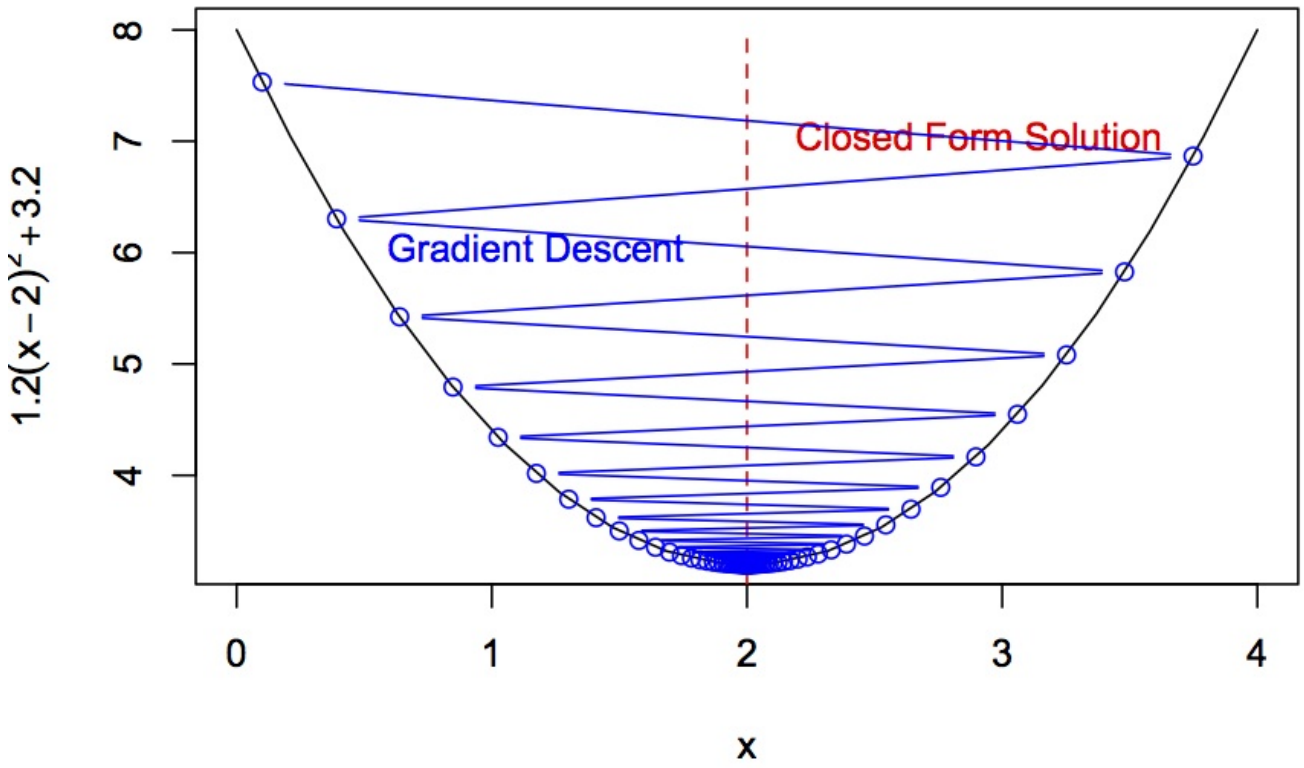
```
#create some values
xs <- seq(0,4,len=20)

#define the function we want to optimize
f<- function(x)
{
    1.2 * (x-2)^2 + 3.2
}

#Plot the function
plot(xs, f(xs), type="l", xlab="x",ylab=expression(1.2 * (x-2)^2 + 3.2))

#Calculate the gradient of the function
grad <- function(x)
{
    1.2*2*(x-2)
}

#Closed form solution
lines(c(2,2), c(3,8), col="red",lty=2)
text(2.1, 7, "Closed Form Solution", col="red", pos=4)

#Implementation of gradient descent
#Initialize the first guess
x<-0.1
#store initial x values
xtrace<-x
#store the y-values
ftrace<-f(x)
# The learning rate alpha
alp <- 0.6
for (step in 1: 100)
{
    #gradient descent step
    x <- x - alp * grad(x)
    xtrace <- c(xtrace,x)
    ftrace <- c(ftrace, f(x))
}
lines(xtrace, ftrace, type = "b", col="blue")
text(0.5, 6, "Gradient Descent", col = "blue", pos=4)
#print final value of x
print(x)

```

Solutions: $\alpha = 0.6$



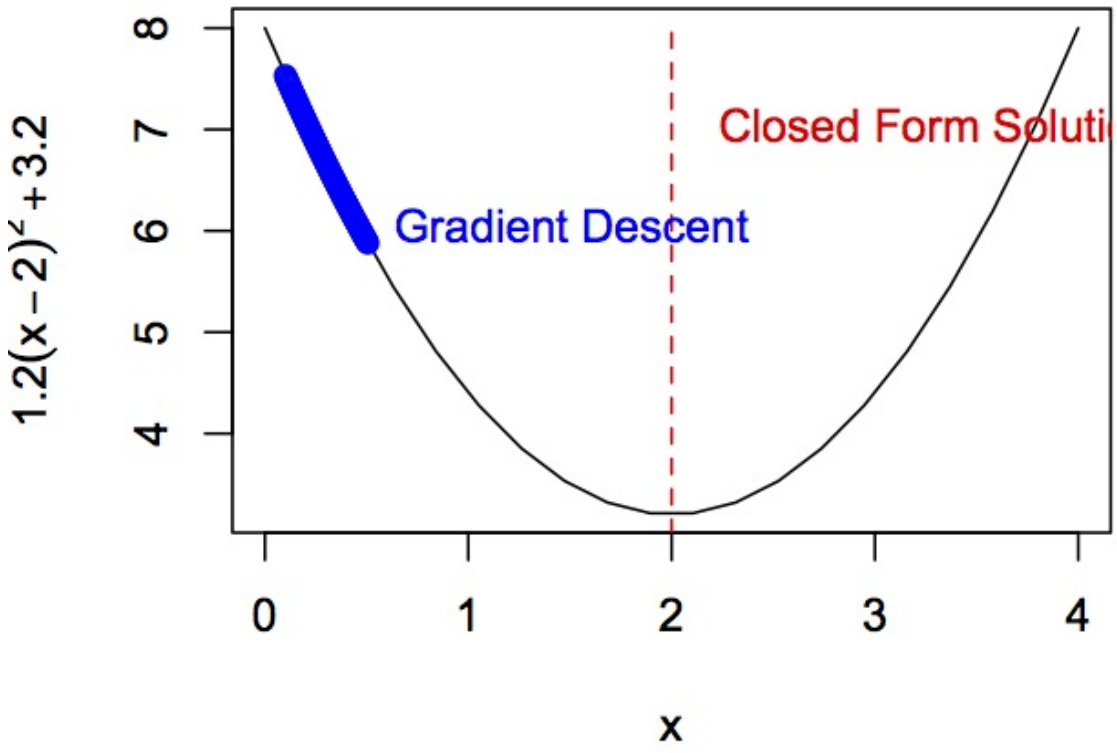
Minima at x=2

Solutions: $\alpha = 0.8$



Minima at x=1.999545

Solutions: $\alpha = 0.001$



Minima at x=0.5058381

Homework

Can you modify the R-code for gradient descent to now perform Newton's Method? Report your result.

Variants of iterative descent algorithms - Scaling

- To improve the rate of convergence of steepest descent method, *scale* the gradient $\nabla f(x_k)$
- Multiply with a positive definite symmetric matrix D_k
- Use $d_k = -D_k \nabla f(x_k)$

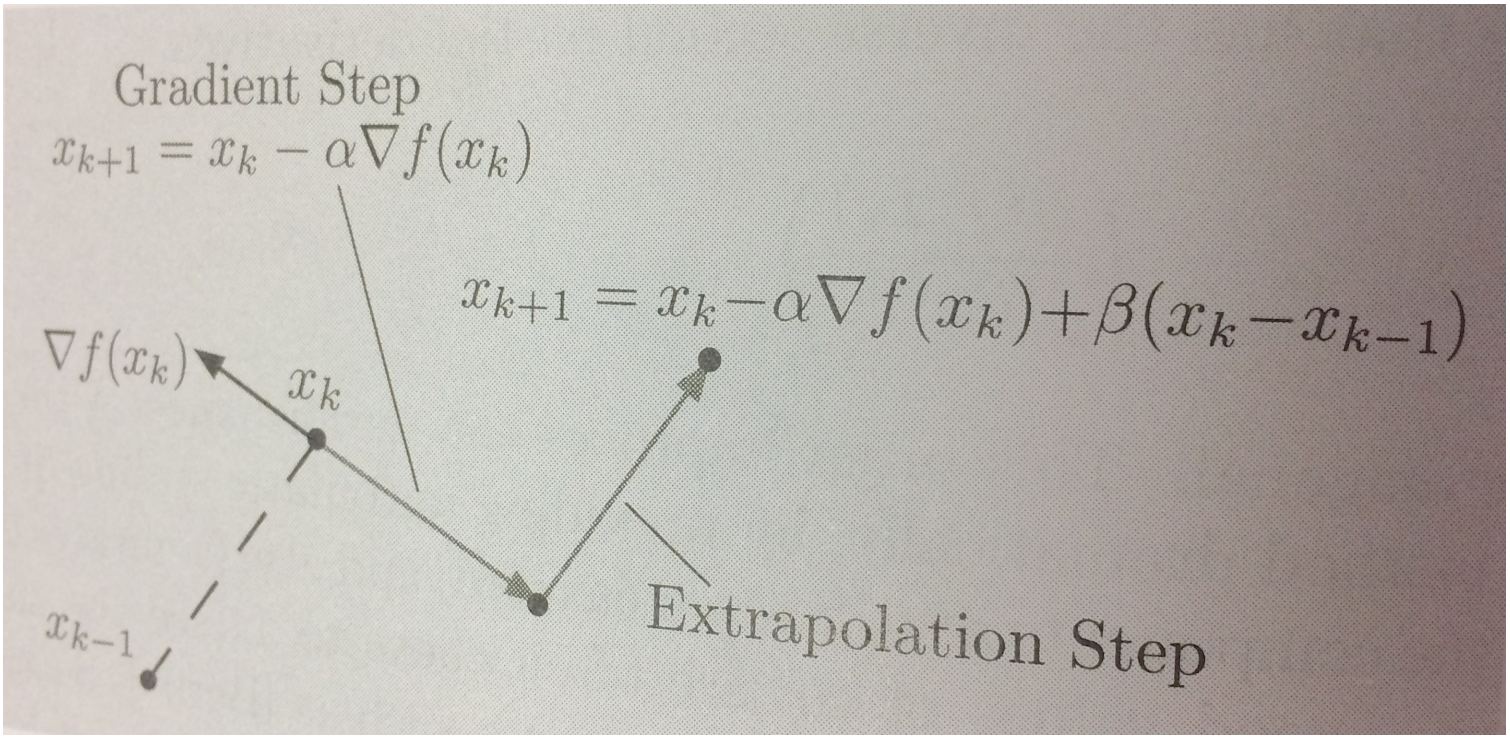
$$x_{k+1} = x_k - \alpha_k D_k \nabla f(x_k)$$

Variants of iterative descent algorithms - Extrapolation

- Gradient Method with momentum
- Extrapolate along the direction of the difference of the preceding two iterates
- Heavy Ball Method: If α_k and β_k are constant
  - With a momentum term, the steepest descent method is less prone to being stuck in a *local* minima
  - Works very well with cost functions that are alternatively very steep and very flat along the path of the algorithm.

$x_{k+1} = x_k - \alpha_k D_k \nabla f(x_k) + \beta_k (x_k - x_{k-1})$
β is a scalar in [0,1)
$x_{-1} = x_0$

Heavy Ball Method



Nesterov's Method

- Two step process
  - Step 1: Extrapolation: $y_k = x_k + \beta_k(x_k - x_{k-1})$, $\beta \to 1$
  - Step 2: Gradient Estimation using constant step size: $x_{k+1} = y_k - \alpha \nabla f(y_k)$
- Optimal iteration complexity properties.
- Comparison to Heavy Ball Method: (1) it reverses order of gradient calculation and extrapolation. (2) Uses $\nabla f(y_k)$ instead of $\nabla f(x_k)$

Conjugate Gradient Method

- Interesting connection between extrapolation method and conjugate gradient method
- If α_k and β_k in an iteration of extrapolation method are chosen optimally for each k, extrapolation reduces to conjugate gradient.

$(\alpha_k, \beta_k) \in argmin_{\alpha,\beta \in R} f(x_k - \alpha_k D_k \nabla f(x_k) + \beta_k (x_k - x_{k-1}), k = 0, 1 \cdots$

Conjugate Gradient Method

- Suppose we want to solve the following system of linear equations $Ax = b$
- A is a symmetric $d \times d$ matrix
- A is positive definite $x^T Ax > 0$ for all non-zero vectors $x \in R^d$
- Conjugate Gradient can be used to solve the problem!
- It extends to a method for minimizing quadratic functions, which can be used to generalize minimization of arbitrary functions $f : R^d \rightarrow R$.

Definition - Conjugate Vectors

- Two vectors are conjugate if and only if they are orthogonal with respect to this inner product.
- Two non-zero vectors u and v are conjugate if $u^T Av = 0$
  - if u is conjugate to v, then v is conjugate to u.
- Suppose that $P = p_1, p_2, \cdots p_d$ is a set of d mutually conjugate vectors, then P forms a basis for R^d
- Solution x^* of $Ax = b$ can be expressed in terms of this basis set.
- $x^* = \sum_{i=1}^{n} \alpha_i p_i$

CG for quadratic functions

- Suppose we have some quadratic function $f(x) = \frac{1}{2}x^T Ax + b^T x + c$, with $x \in R^d$, $A \in R^{d \times d}$, $b \in R^d$, $c \in R$.
- Taking gradient of f, $\nabla f(x) = Ax + b$.
- If we evaluate $-\nabla f(x)$ at any given location, it will give us a vector pointing towards the direction of steepest descent.

A Proposition

- Pick some initial x_0
- Compute the gradient $-\nabla f(x_0)$
- Move in that direction by some step size α
- Unlike normal gradient descent, we do not have a fixed step size α

Perform a line search in order to find the *best* α

- This α is the value of α which brings us to the minimum of $f(x)$ if we are constrained to move in the direction given by $d_0 = -\nabla f(x_0)$.

Compute α

$$\begin{aligned} g(\alpha) &= f(x_0 + \alpha d_0) \\ &= \frac{1}{2}(x_0 + \alpha d_0)^T A(x_0 + \alpha d_0) + b^T(x_0 + \alpha d_0) + c \end{aligned}$$

- Minimum of this function can be found by $g'(\alpha) = 0$.
- Solve for α. $\alpha = -\frac{d_i^T(Ax_i + b)}{d_i^T A d_i}$

An important observation

- If this were simple gradient descent, we would compute the gradient at each next point and move in that direction.
- By moving α_0 in direction d_0 and then α_1 in direction d_1, can we ruin all the work done to find the optimal α?
- Rectify this, by requiring that our directions be *conjugate* to one another.

Which direction to move?

- We have already moved in the direction $d_0 = -\nabla f(x_0)$.
- Find direction d_1 conjugate to d_0. How?
- Compute d_1 at x_1. Subtract anything that would counter-act the previous direction.
- $d_1 = -\nabla f(x_0) + \beta_0 d_0$. What is β_0?
- Since d_0 and d_1 are conjugate, $d_1^T A d_0 = 0$

$\beta_0 = \frac{\nabla f(x_1)^T A d_0}{d_0^T A d_0}$.
Choosing this β gives us a direction conjugate to all previous directions.
Iterating this will keep giving us conjugate directions.

Stochastic Gradient Descent

- Minimize R_n
- Start with an initial $w_1 \in R^d$
- Compute the gradient using the formula: $w_{k+1} = w_k - \nabla f_{i_k}(w_k)$
- Index i_k is chosen randomly from $1, 2, \cdots, n$
- Each iteration of this method is computationally cheap - gradient is estimated on only ONE sample!

Comments on the Stochastic Gradient Descent

- $\{w_k\}$ is a stochastic process whose behavior is determined by the random sequence $\{i_k\}$
- Each direction $-\nabla f_{i_k}(w_k)$ might not be a descent direction from w_k
- In expectation, if the above is a descent direction, then the sequence can be guided towards the minimizer of R_n

Outline

The Projection Method

- A variant of the gradient descent method, also called *gradient projection method*.
- $P_X()$ denotes a projection on X, $\alpha_k > 0$ is the step size.

$$x_{k+1} = P_X(x_k - \alpha_k \nabla f(x_k))$$

Drawbacks of the Projection Method

- It's rate of convergence is often similar to steepest descent and therefore often slow.
  - Can be overcome by scaling techniques.
- Depending on the nature of X, the projection operation may involve substantial overhead.

Coordinate Descent

Is it possible to find a descent approach where the gradient and the Hessian of the function will not be estimated?

The Intuition

- Iterative Method
- Obtain an iterate as follows: Fix components of variable x at their values at current iteration
- Approximately minimize the objective with respect to remaining components.
- Each subproblem is a lower dimensional minimization problem
- Can be solved more easily than the overall problem.

The Algorithm

- Set $k \rightarrow 0$

The Algorithm

- Set $k \rightarrow 0$
- Choose $x^0 \in R^n$

The Algorithm

- Set $k \to 0$
- Choose $x^0 \in R^n$
- repeat
  - Choose index $i \in \{1, 2, \cdots, n\}$
  - Optimize the i^{th} coordinate
    $x_i^{k+1} \to \text{argmin}_{\xi \in R} f(\underbrace{x_1^{k+1}, \cdots x_{i-1}^{k+1}}_{\text{done}}, \underbrace{\xi}_{\text{current solution}}, \underbrace{x_{i+1}^{k}, \cdots x_n^{k}}_{\text{todo}})$
  - $k \to k + 1$

The Algorithm

- Set $k \to 0$
- Choose $x^0 \in R^n$
- repeat
  - Choose index $i \in \{1, 2, \cdots, n\}$
  - Optimize the i^{th} coordinate
    $x_i^{k+1} \to \text{argmin}_{\xi \in R} f(\underbrace{x_1^{k+1}, \cdots x_{i-1}^{k+1}}_{\text{done}}, \underbrace{\xi}_{\text{current solution}}, \underbrace{x_{i+1}^{k}, \cdots x_n^{k}}_{\text{todo}})$
  - $k \to k + 1$
- Decide when/how to stop.

The Algorithm

- Set $k \to 0$
- Choose $x^0 \in R^n$
- repeat
  - Choose index $i \in \{1, 2, \cdots, n\}$
  - Optimize the i^{th} coordinate
    $x_i^{k+1} \to \text{argmin}_{\xi \in R} f(\underbrace{x_1^{k+1}, \cdots x_{i-1}^{k+1}}_{\text{done}}, \underbrace{\xi}_{\text{current solution}}, \underbrace{x_{i+1}^{k}, \cdots x_n^{k}}_{\text{todo}})$
  - $k \to k + 1$
- Decide when/how to stop.
- Return x^k

Which index should be chosen?

- Gauss Southwell: If the function is differentiable, at iteration k, pick the index that minimizes $[\nabla f(x_k)]_i$
- Derivative Free Rules:
  - Cyclic Order: $1, 2, \cdots, n, 1, \cdots$
  - Almost Cyclic: Each coordinate $1 \leq i \leq n$ will be picked at least once every B iterations, $B \geq n$.
  - Double Sweep: $1, \cdots, n, n, \cdots, 1$, repeat.
  - Cyclic with permutation: Random order each cycle.
  - Random Sampling: Pick a random index at each iteration.

Block Coordinate Descent

minimize f(x)
subject to $x \in X$ where
$f : R^n \rightarrow R$ is a differentiable function
X is a Cartesian product of closed convex sets $X_1, X_2, \cdots, X_m$
$X = X_1 \times X_2 \times \cdots X_m$

Block Coordinate Descent

- The vector x is partitioned as $x = (x^1, x^2, \cdots, x^m)$, where each x^i belongs to R^{n_i}
- The constraint $x \in X$ is equivalent to $x^i \in X_i, i = 1, \cdots, m$
- When $n_i = 1$ for all i, x^is are scalars.
- Minimize with respect to a single component x^i at each iteration with all the other components fixed.

Block Coordinate Descent

$$x_k = (x_k^1, \cdots, x_k^m)$$
$$x_{k+1} = (x_{k+1}^1, \cdots, x_{k+1}^m)$$
$$x_{k+1}^i \in \arg\min f(x_{k+1}^1, \cdots, x_{k+1}^{i-1}, \xi, x_k^{i+1}, \cdots, x_k^m), i = 1, \cdots, m$$

- At each iteration the cost is minimized with respect to each of the "block coordinate" vectors x^i taken one at a time in cyclic order.
- Makes sense only if the minimization can be performed fairly easily.
- When x^i is scalar this is straightforward.

Outline

A Little Digression - Subgradients of convex functions

Definition: A vector $g \in R^d$ is a *subgradient* of $f : R^d \rightarrow R$ at $x \in$ **dom** f if for all $z \in$ **dom** f

$$f(z) \geq f(x) + g^T(z - x)$$

- If f is convex and differentiable, then its gradient at x is a sub gradient.
- BUT, sub gradients exist even for **non-differentiable** functions.

Subgradients - An illustration

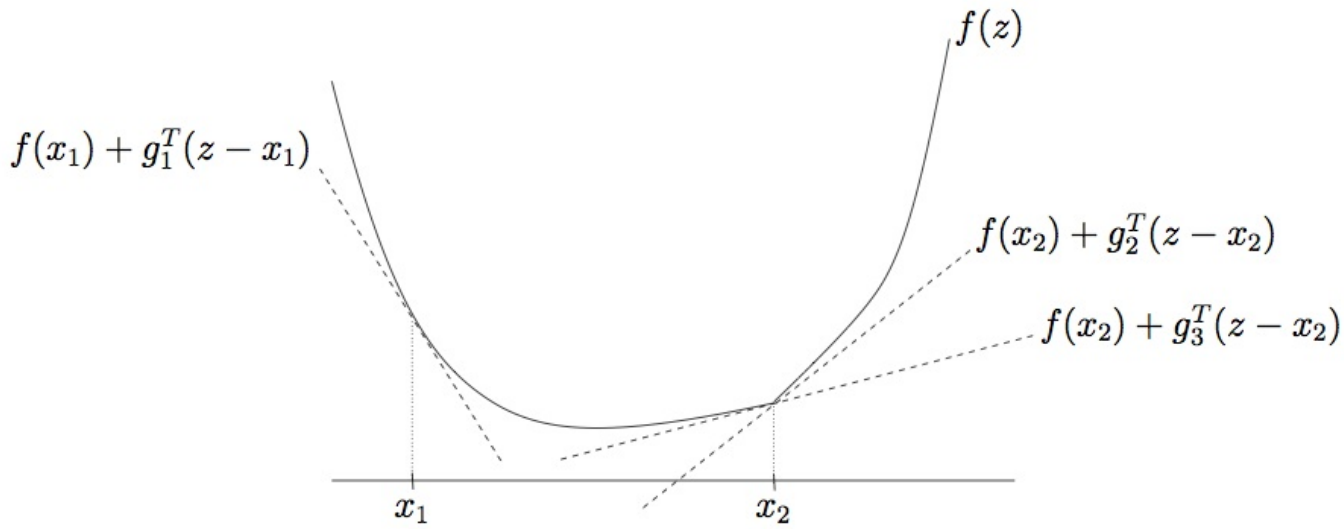


Figure: At x_1 the convex function f is differentiable, and g_1 (which is the derivative of f at x_1) is the unique sub gradient at x_1. At the point x_2, f is not differentiable. At this point, f has many sub gradients: two sub gradients - g_2 and g_3 are shown here.
Figure adapted from `https://see.stanford.edu/materials/lsocoee364b/01-subgradients_notes.pdf`

Subgradients - An Example

Absolute Value: Consider $f(z) = |z|$. For $x < 0$ the sub gradient is unique $\partial f = -1$. Similarly, for $x > 0$ we have $\partial f = 1$. What happens at $x = 0$?
At $x = 0$, the sub differential is defined by the inequality $|z| \geq gz$ for all z, which is satisfied if and only if $g \in [-1, 1]$. Therefore, we have $\partial f(0) = [-1, 1]$.

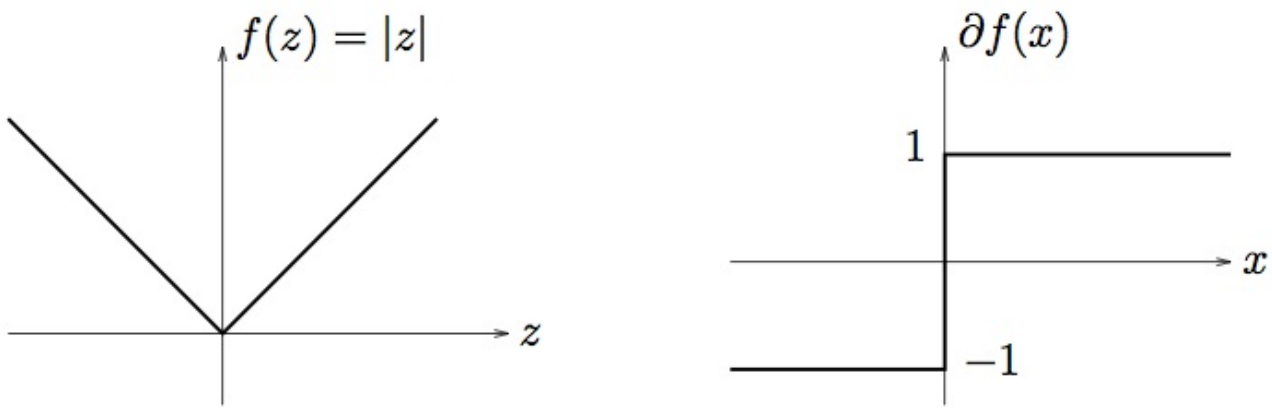


Figure: The absolute value function (left) and its sub differential $\partial f(x)$ as a function of x (right).

Polyhedral Approximation

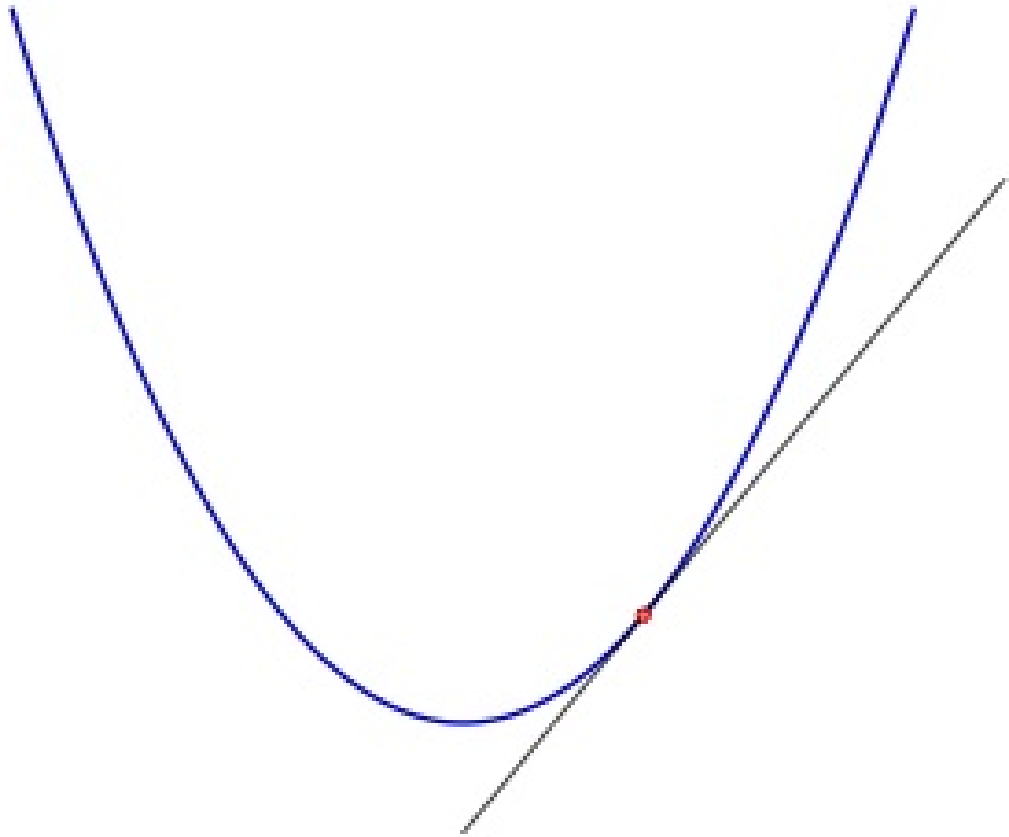
- Generate a sequence x_k by solving at each k an approximate problem

$$x_{k+1} \in \underset{x \in X_k}{\operatorname{argmin}} F_k(x)$$

where,

- F_k is a polyhedral function that approximates f
- X_k is a polyhedral set that approximates X
- Assumption: The polyhedral structure of the problem is easier to solve than the original.

First Order Taylor Approximation



For any x and x' we have $f(x) \geq f(x') + \langle x - x', \nabla f(x') \rangle$

Cutting Plane Methods or Localization Methods

Cutting planes [SZV09, TVSL10] are hyperplanes that separate the current point from the optimal points.

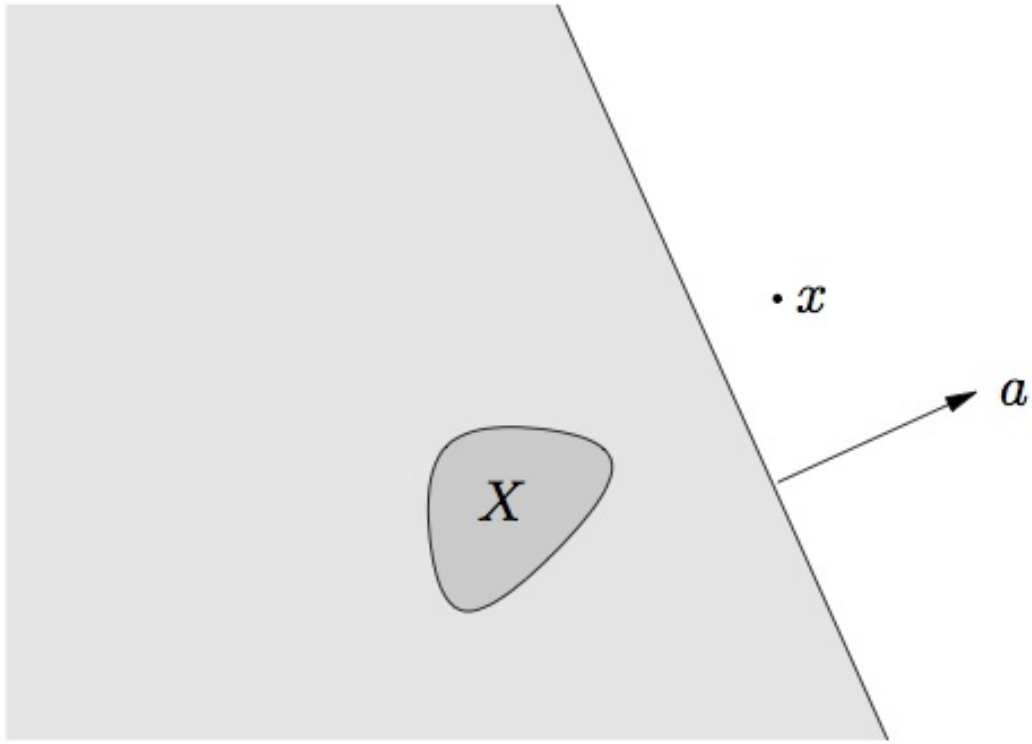


Figure: The inequality $a^T x \leq b$ defines a cutting plane at the the query point x, for the target set X. To find a point in a target set we need only search the lightly shaded half-space; the unshaded half-space $\{z|a^T z > b\}$ cannot contain any point in the target set.

An Intuition

- In cutting plane methods, we do not have direct access to any description of the target set X except through an oracle.
- Query oracle at point $x \in R^d$
- Oracle returns the following information: Either $x \in X$, in which case we are done.
- OR Return a separating hyperplane between x and X
- The separating hyperplane is also called a **CUT**. It eliminates a half space from our search.

Neutral and Deep Cuts

- **Neutral Cut:** When the cutting plane $a^T z = b$ contains the query point x, it is a neutral cut.
- **Deep Cut:** When $a^T x > b$, i.e. x lies in the interior of the half space that is being cut from consideration, the cutting plane is called a deep cut.

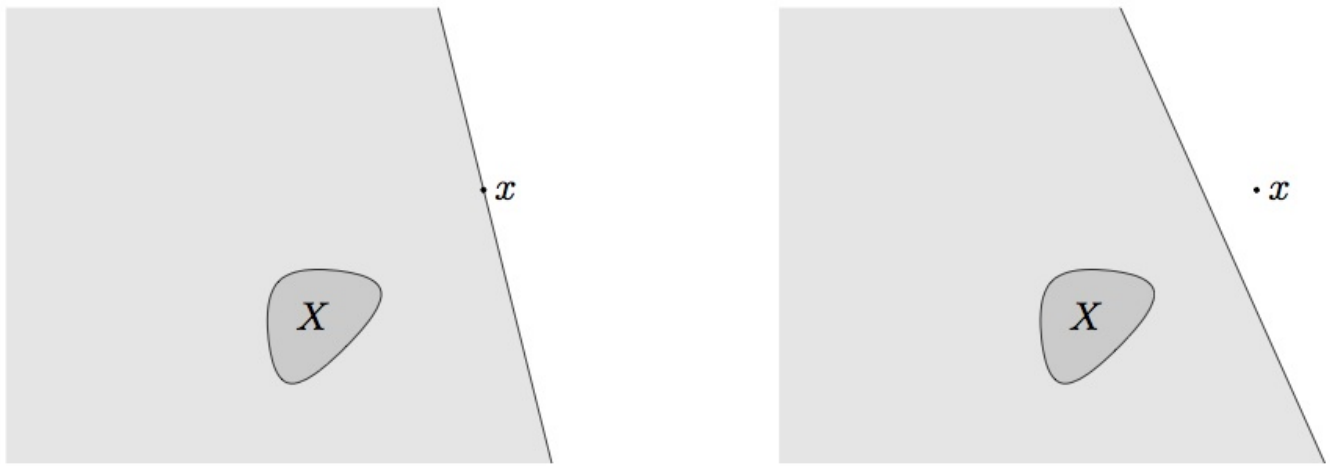


Figure: (Left) A neutral cut. (Right) A deep cut.

Why use Cutting Plane Methods?

- Cutting-plane methods do not require differentiability of the objective and constraint functions. Each iteration requires the computation of a sub gradient of the objective or constraint functions.
- Cutting-plane methods can exploit certain types of structure in large and complex problems.
- Cutting-plane methods do not require evaluation of the objective and all the constraint functions at each iteration. This can make cutting-plane methods useful for problems with a very large number of constraints.

The Goal

- Find a point in a convex set (also called a **target** set) $X \subseteq R^d$
- Target set X can be taken as the set of optimal points for the problem
- Goal: Find an optimal point for the optimization problem

Illustration - Cutting Plane Method

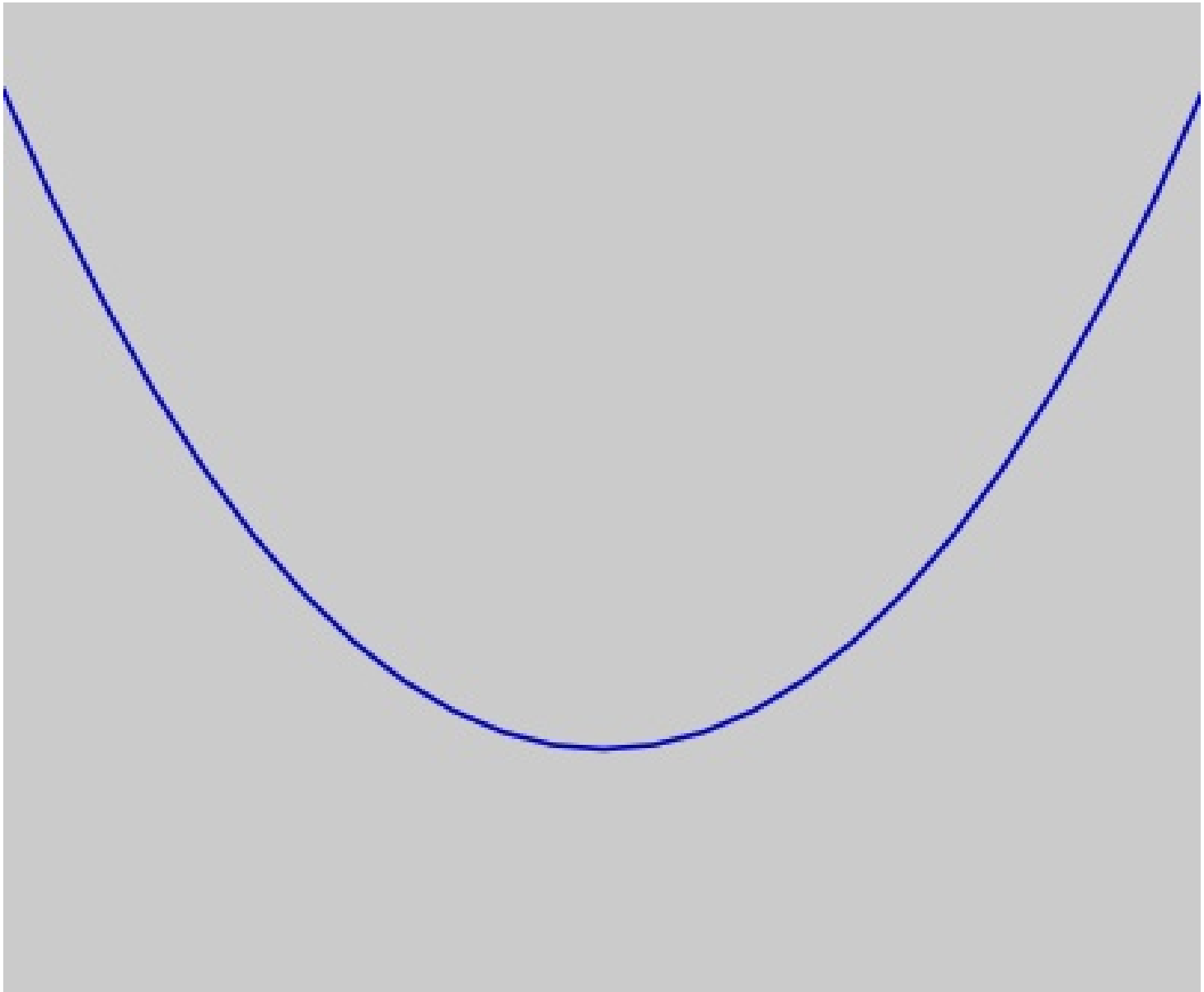


Illustration - Cutting Plane Method

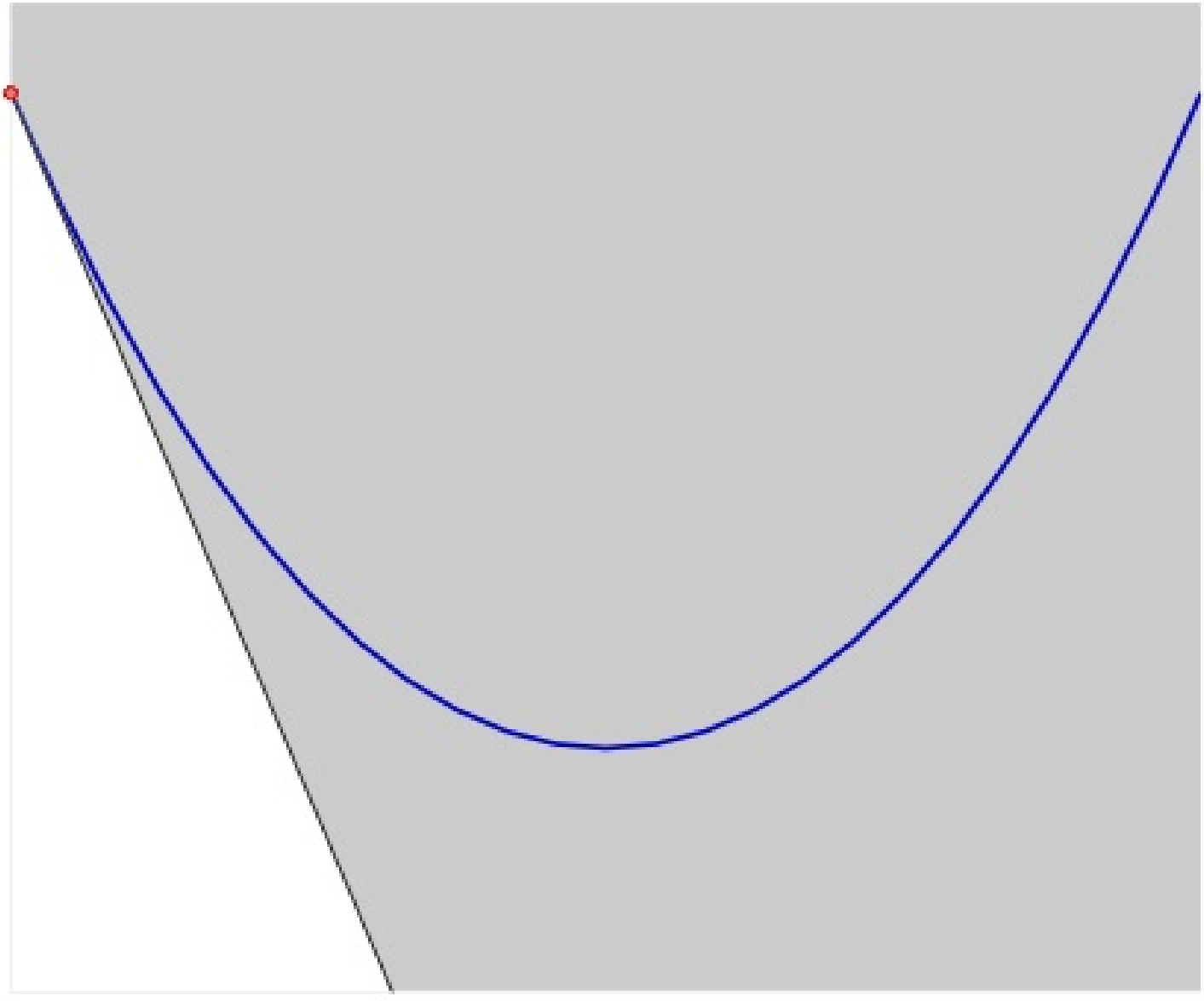


Illustration - Cutting Plane Method

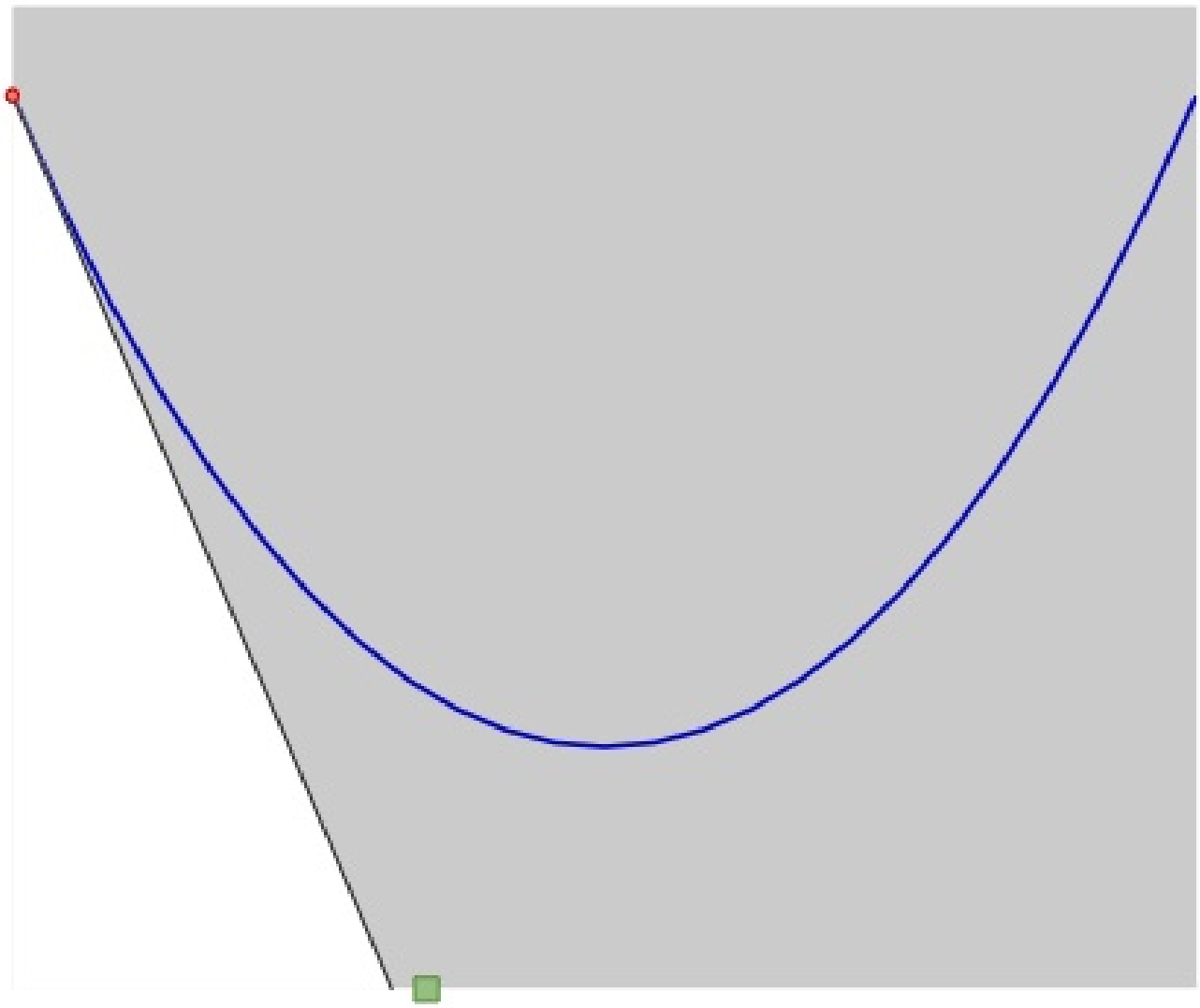


Illustration - Cutting Plane Method

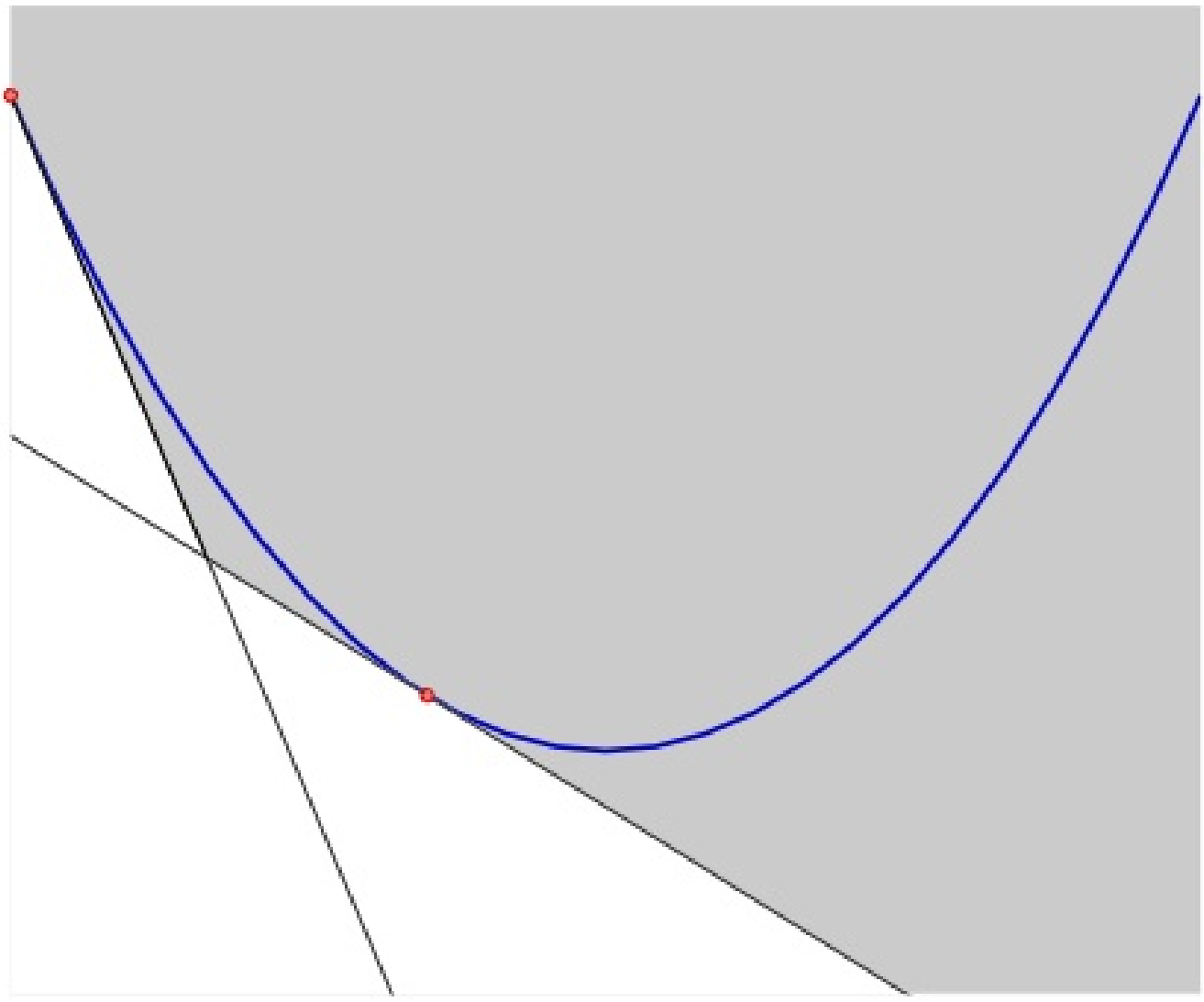


Illustration - Cutting Plane Method

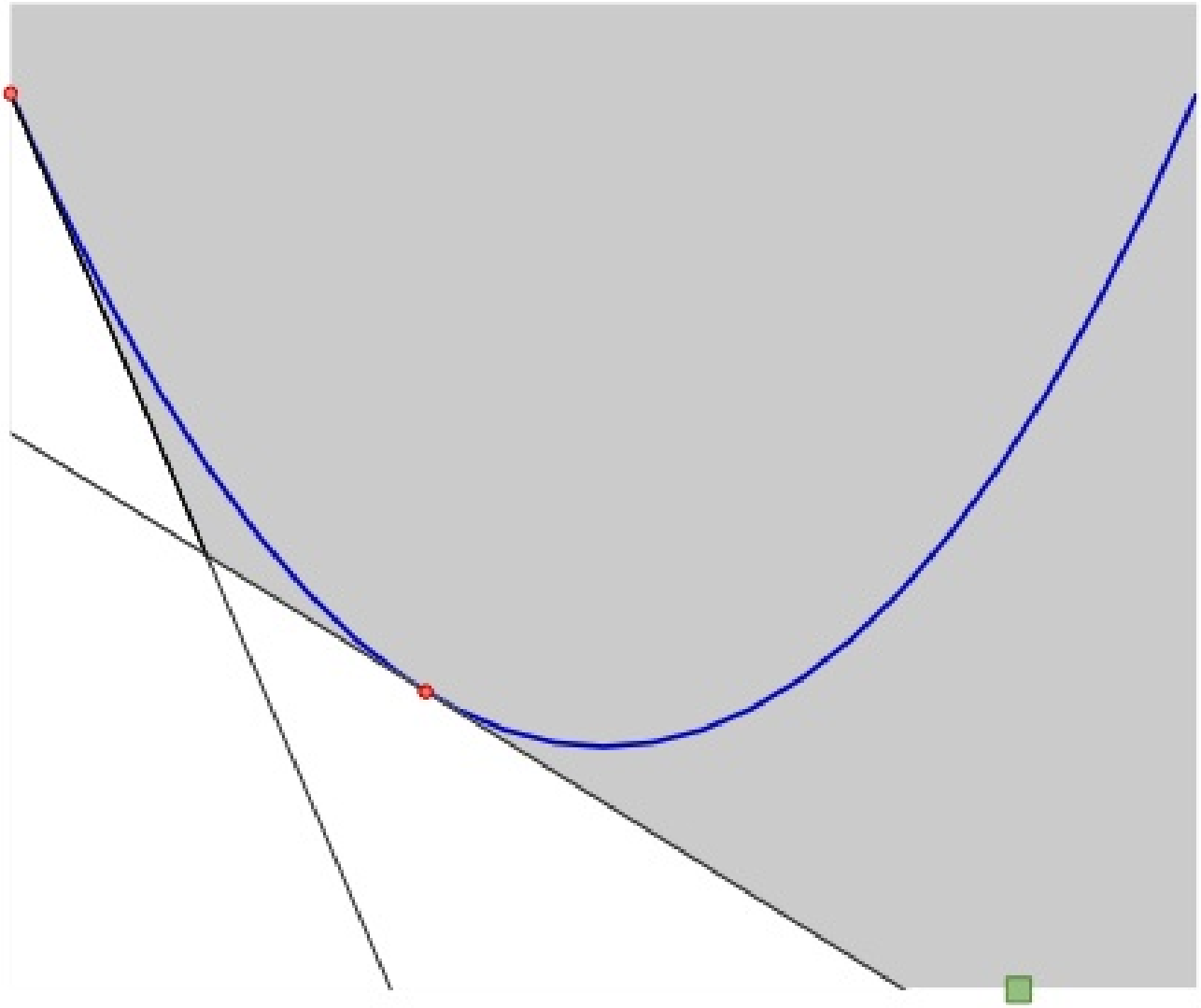


Illustration - Cutting Plane Method

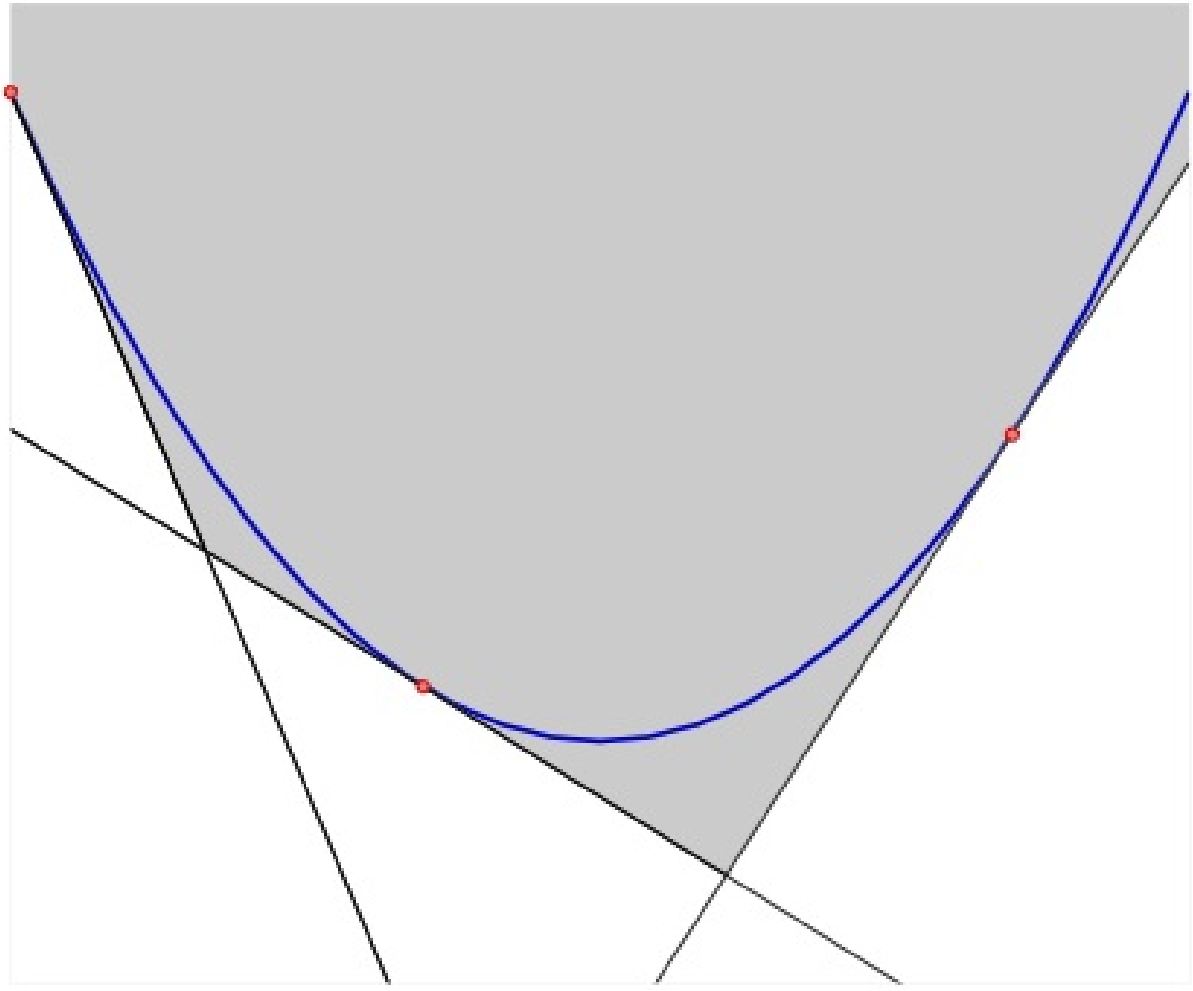


Illustration - Cutting Plane Method

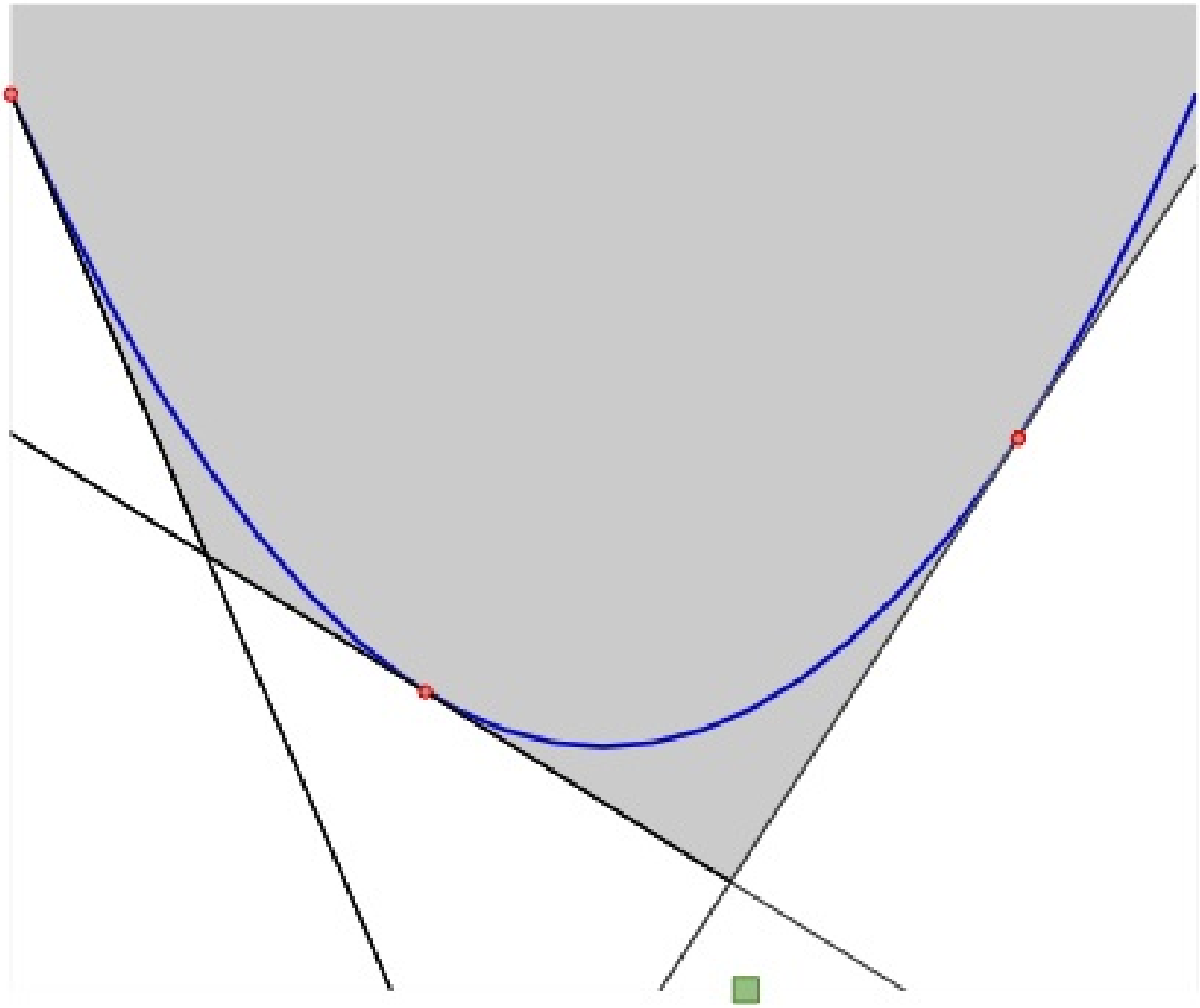


Illustration - Cutting Plane Method

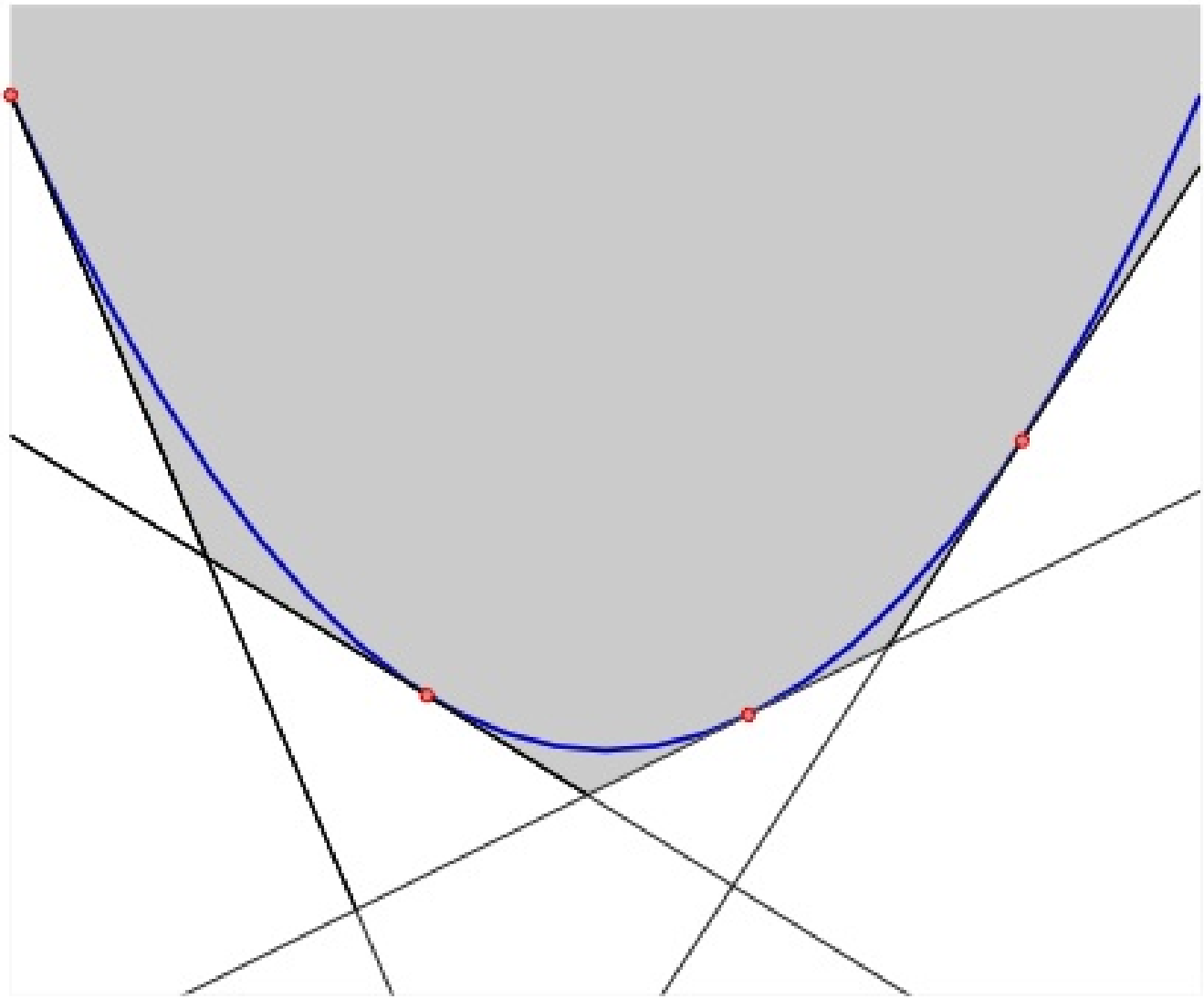
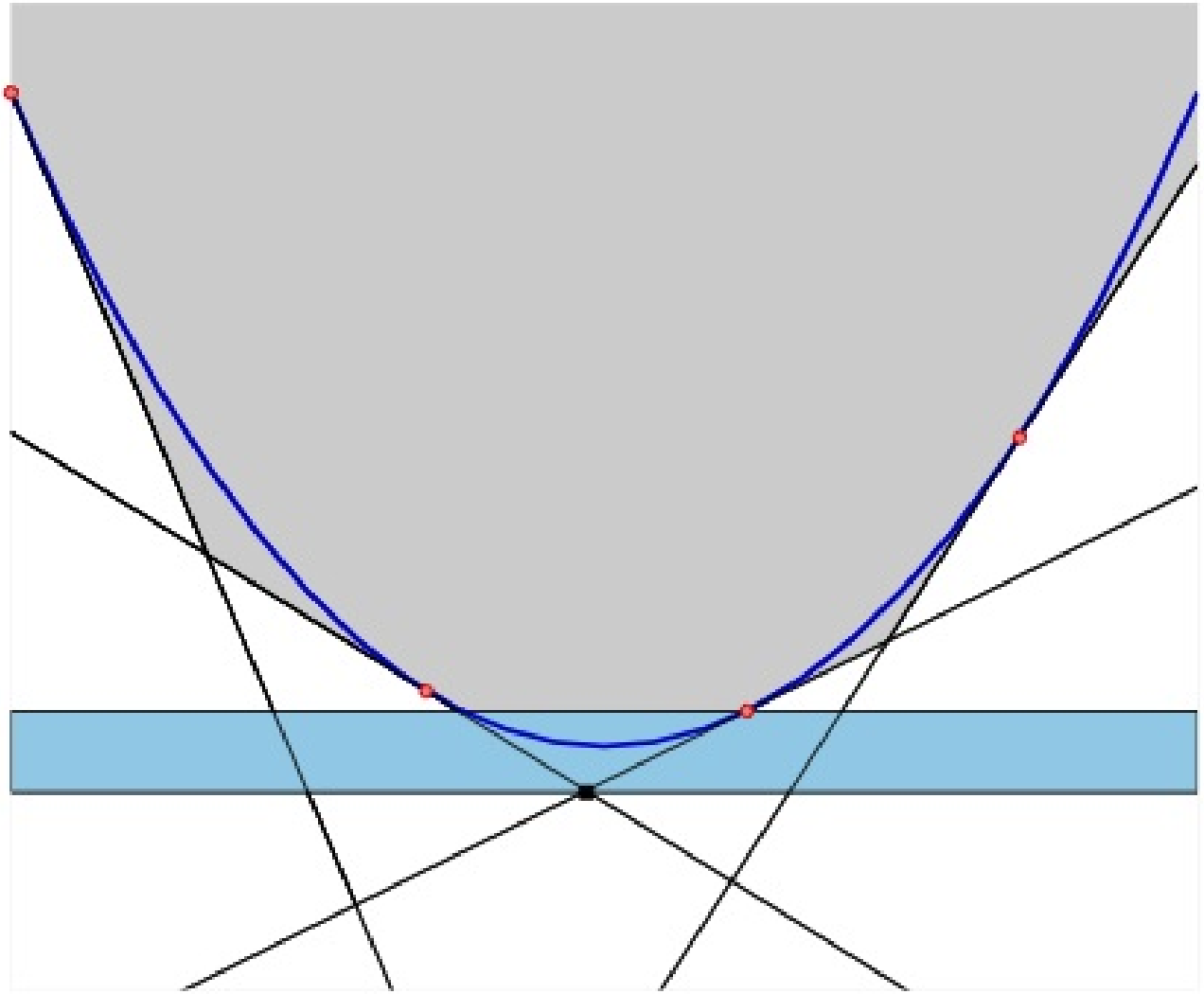


Illustration - Cutting Plane Method



The set-up

- Solve the optimization problem $\min \{f(x) | x \in X\}$ where $X \subseteq R^d$ is a convex set, $f : R^d \to R$ is a convex function and a minimum exists.
- Set X can be accessed by a *separation oracle*
- Given $\hat{x} \in X$, the separation oracle says either (1) $\hat{x} \in X$ or (2) Return a hyperplane, *cutting plane* $\langle a, x \rangle \leq b$ that separates $\hat{x}$ from X

The Cutting Plane Algorithm

- **Initialization:** $t \leftarrow 0$, $X_0 \supseteq X$

The Cutting Plane Algorithm

- **Initialization:** $t \leftarrow 0$, $X_0 \supseteq X$
- **loop**

The Cutting Plane Algorithm

- **Initialization:** $t \leftarrow 0$, $X_0 \supseteq X$
- **loop**
- Let $x_t \in \text{argmin}_{x \in X_t}$ f(x)

The Cutting Plane Algorithm

- **Initialization:** $t \leftarrow 0$, $X_0 \supseteq X$
- **loop**
- Let $x_t \in \text{argmin}_{x \in X_t} \text{f(x)}$
- If $x_t \in X$ then stop. Else find a cutting plane $\langle a, x \rangle \leq b$ separating x_t from X

The Cutting Plane Algorithm

- **Initialization:** $t \leftarrow 0$, $X_0 \supseteq X$
- **loop**
- Let $x_t \in \text{argmin}_{x \in X_t} \text{f(x)}$
- If $x_t \in X$ then stop. Else find a cutting plane $\langle a, x \rangle \leq b$ separating x_t from X
- $X_{t+1} \rightarrow X_t \cap \{x | \langle a, x \rangle \leq b\}$

The Cutting Plane Algorithm

- **Initialization:** $t \leftarrow 0$, $X_0 \supseteq X$
- **loop**
- Let $x_t \in \text{argmin}_{x \in X_t} \text{f(x)}$
- If $x_t \in X$ then stop. Else find a cutting plane $\langle a, x \rangle \leq b$ separating x_t from X
- $X_{t+1} \rightarrow X_t \cap \{x | \langle a, x \rangle \leq b\}$
- $t \leftarrow t + 1$

The Cutting Plane Algorithm

- **Initialization:** $t \leftarrow 0$, $X_0 \supseteq X$
- **loop**
- Let $x_t \in \text{argmin}_{x \in X_t}$ f(x)
- If $x_t \in X$ then stop. Else find a cutting plane $\langle a, x \rangle \leq b$ separating x_t from X
- $X_{t+1} \rightarrow X_t \cap \{x | \langle a, x \rangle \leq b\}$
- $t \leftarrow t + 1$
- **end loop**

The Trick

Approximate X well by a convex polyhedron but only *near the optimum*. This is best seen if X is already a convex polyhedron , described by a set of linear inequalities. At optimum only some of the inequalities are active. Remove all inactive inequalities - Problem unaffected!

Outline

Other Uses of Optimization

- Support Vector Machine (SVM) formulations for classification, regression, ranking, and novelty detection require the solutions of large dense QPs or LPs.
- Belief Propagation
- Large scale Multi-Kernel Learning
- Ranking Algorithms that exploit structure
- Max Margin Methods for Structured Output

Conclusion

- Several iterative descent techniques (including gradient descent, newton method, conjugate gradient method, SGD, coordinate descent and others)
- Examined polyhedral approximation using cutting plane methods
- Some implementations of the above techniques are available from the github repo.
- The MOSEK interface to R and RMOSEK package has some nice implementations of these techniques
- We did not cover many things - convex optimization in active learning, non-convex optimization, and a lot more!

Some Useful References

- Software
  - The RMOSEK Package: `http://rmosek.r-forge.r-project.org/`
  - CVX: Matlab software for disciplined convex optimization `http://cvxr.com/cvx/`
  - CPLEX: `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`
- MOOC on Convex Optimization: `https://lagunita.stanford.edu/courses/Engineering/CVX101/Winter2014/about`

Interested in Convex Optimization Methods for ML?

If you are a

- Ph.D. student interested in applying convex optimization to your application
- Interested in collaborating on research - please visit `www.buffalo.edu/~haimonti`

Please drop me an email: `haimonti@buffalo.edu`

References

[DPL+11] Haimonti Dutta, Rebecca J Passonneau, Austin Lee, Axinia Radeva, Boyi Xie, and David Waltz.
Learning parameters of the k-means algorithm from subjective human annotation.
In *Twenty-Fourth International FLAIRS Conference*, 2011.

[SBB12] Amarjot Singh, Ketan Bacchuwar, and Akshay Bhasin.
A survey of ocr applications.
International Journal of Machine Learning and Computing, 2(3):314–318, 2012.

[SZV09] Ankan Saha, Xinhua Zhang, and S. V. N. Vishwanathan.
Lower bounds for BMRM and faster rates for training svms.
CoRR, abs/0909.1334, 2009.

[TVSL10] Choon Hui Teo, S.V.N. Vishwanthan, Alex J. Smola, and Quoc V. Le.
Bundle methods for regularized risk minimization.
J. Mach. Learn. Res., 11:311–365, March 2010.