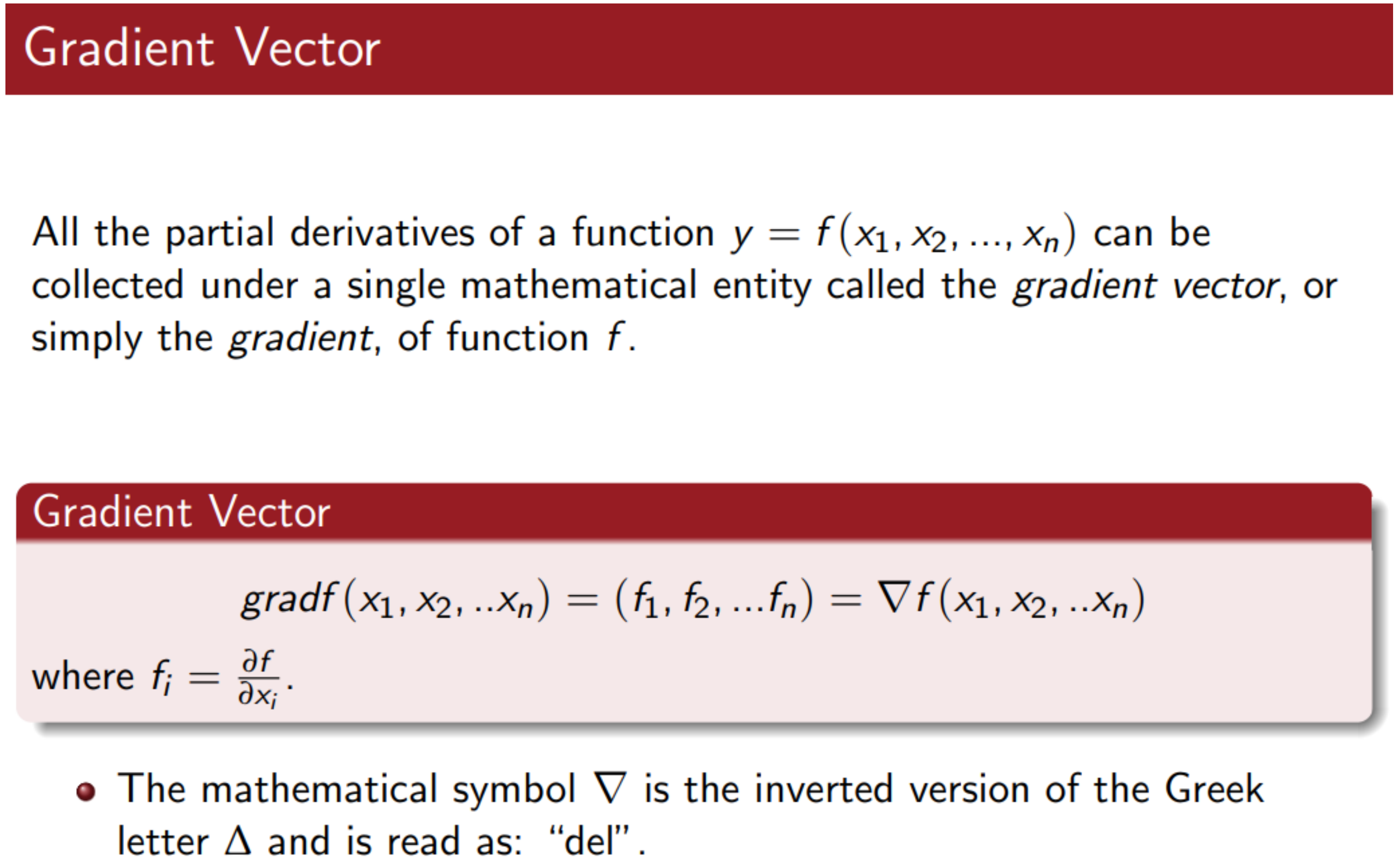
ILLINOIS INSTITUTE OF TECHNOLOGY

Introduction to 1st Order and Convex Optimization Methods in Linear Programming application

Nathan Cao (MSC-514)
Professor Priyanka Sharma
April 8th, 2021

First order and second order optimization application

Firstly, review of optimization knowledge learnt from last semester
– Gradient Vector



Gradient Vector

All the partial derivatives of a function $y = f(x_1, x_2, ..., x_n)$ can be collected under a single mathematical entity called the *gradient vector*, or simply the *gradient*, of function f.

Gradient Vector

$$gradf(x_1, x_2, ..x_n) = (f_1, f_2, ...f_n) = \nabla f(x_1, x_2, ..x_n)$$

where $f_i = \frac{\partial f}{\partial x_i}$.

- The mathematical symbol ∇ is the inverted version of the Greek letter Δ and is read as: “del”.

First order and second order optimization application

Firstly, review of optimization knowledge learnt from last semester
- Higher Order derivatives

Higher order partial derivatives

- **Second partial derivatives**

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_i}\right)$$

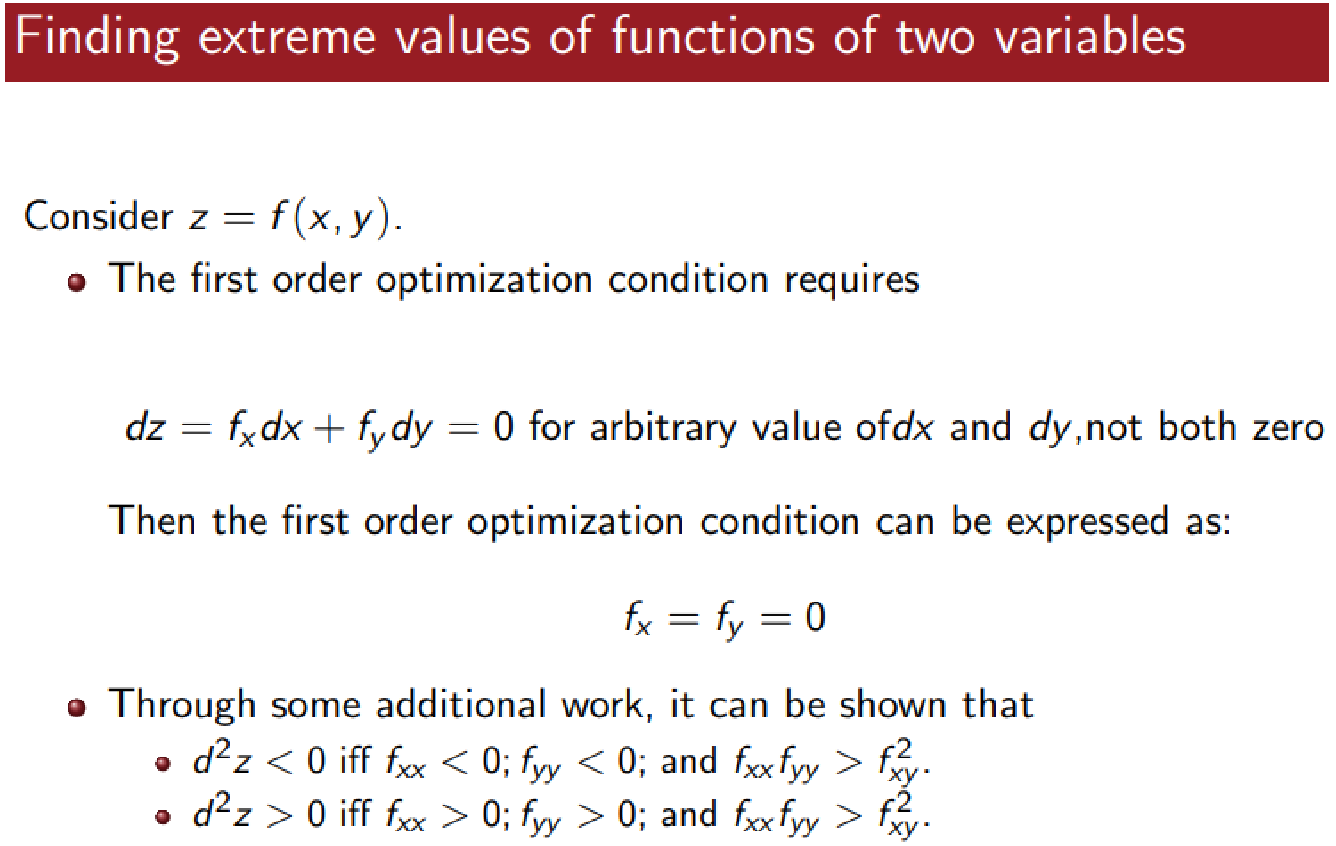
- **Mixed partial derivatives**

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right)$$

for $i \neq j$.

First order and second order optimization application

Firstly, review of optimization knowledge learnt from last semester
- Extreme value: optimal maximizer, minimizer or inflection point



Finding extreme values of functions of two variables

Consider $z = f(x, y)$.

- The first order optimization condition requires

  $$dz = f_x dx + f_y dy = 0 \text{ for arbitrary value of} dx \text{ and } dy \text{,not both zero}$$

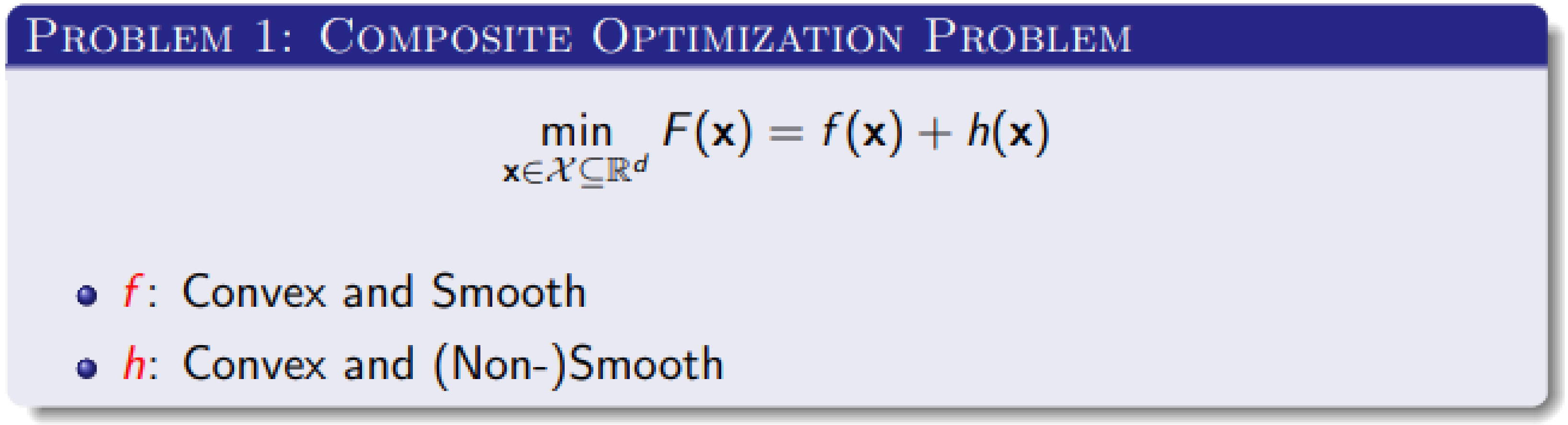
  Then the first order optimization condition can be expressed as:

  $$f_x = f_y = 0$$

- Through some additional work, it can be shown that
  - $d^2z < 0$ iff $f_{xx} < 0$; $f_{yy} < 0$; and $f_{xx}f_{yy} > f_{xy}^2$.
  - $d^2z > 0$ iff $f_{xx} > 0$; $f_{yy} > 0$; and $f_{xx}f_{yy} > f_{xy}^2$.

First order and second order optimization application

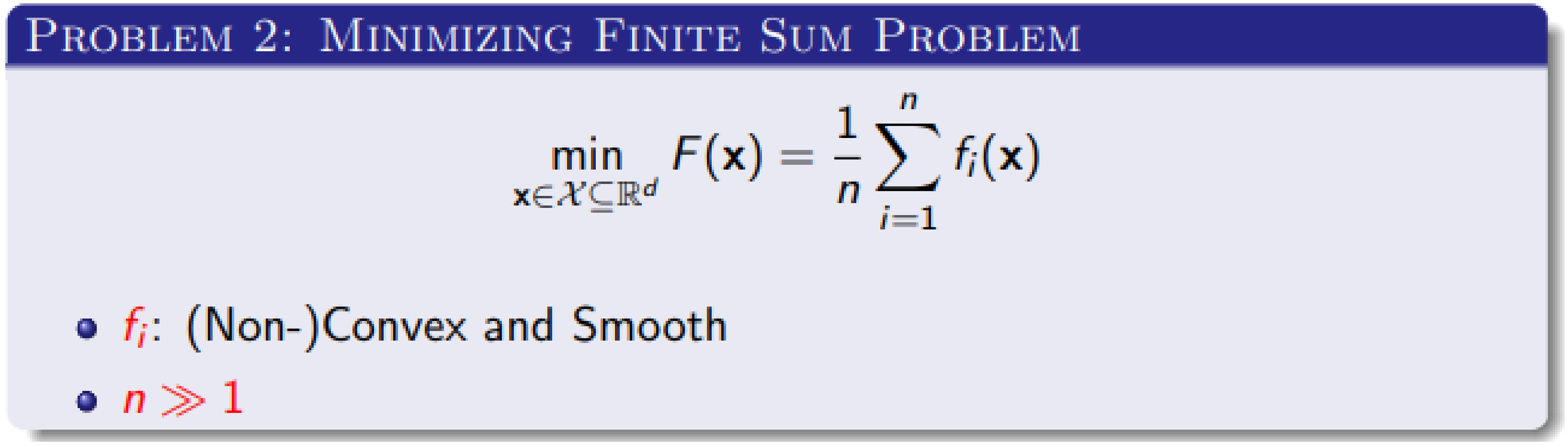
In practical operation use cases, we usually ask how to calculate minimum or maximum value given a series of inputs, these inputs can be serial and/or stochastic



PROBLEM 1: COMPOSITE OPTIMIZATION PROBLEM

$$\min_{\mathbf{x}\in\mathcal{X}\subseteq\mathbb{R}^d} F(\mathbf{x}) = f(\mathbf{x}) + h(\mathbf{x})$$

- f: Convex and Smooth
- h: Convex and (Non-)Smooth



PROBLEM 2: MINIMIZING FINITE SUM PROBLEM

$$\min_{\mathbf{x}\in\mathcal{X}\subseteq\mathbb{R}^d} F(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$$

- f_i: (Non-)Convex and Smooth
- $n \gg 1$

First order and second order optimization application

To solve optimization question, start with first order and second order optimization technologies are good options.

First Order Methods

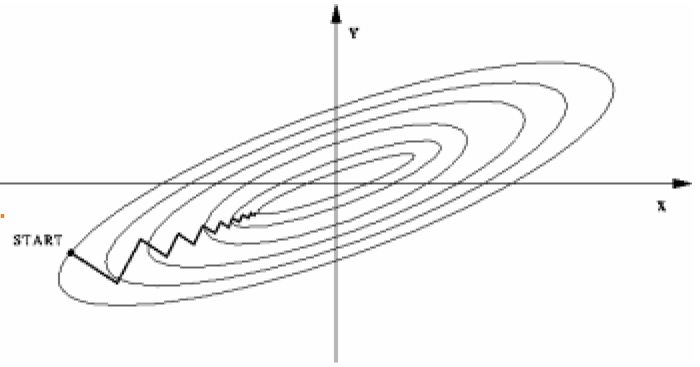
- Variants of Gradient Descent (GD):
  - Reduce the per-iteration cost of GD ⇒ Efficiency
  - Achieve the convergence rate of the GD ⇒ Effectiveness

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \nabla F(\mathbf{x}^{(k)})$$

- E.g.: SAG, SDCA, SVRG, Prox-SVRG, Acc-Prox-SVRG, Acc-Prox-SDCA, S2GD, mS2GD, MISO, SAGA, AMSVRG, ...

Q: Why do we use (stochastic) 1st order method?

- Cheaper Iterations? i.e., $n \gg 1$ and/or $d \gg 1$
- Avoids Over-fitting?



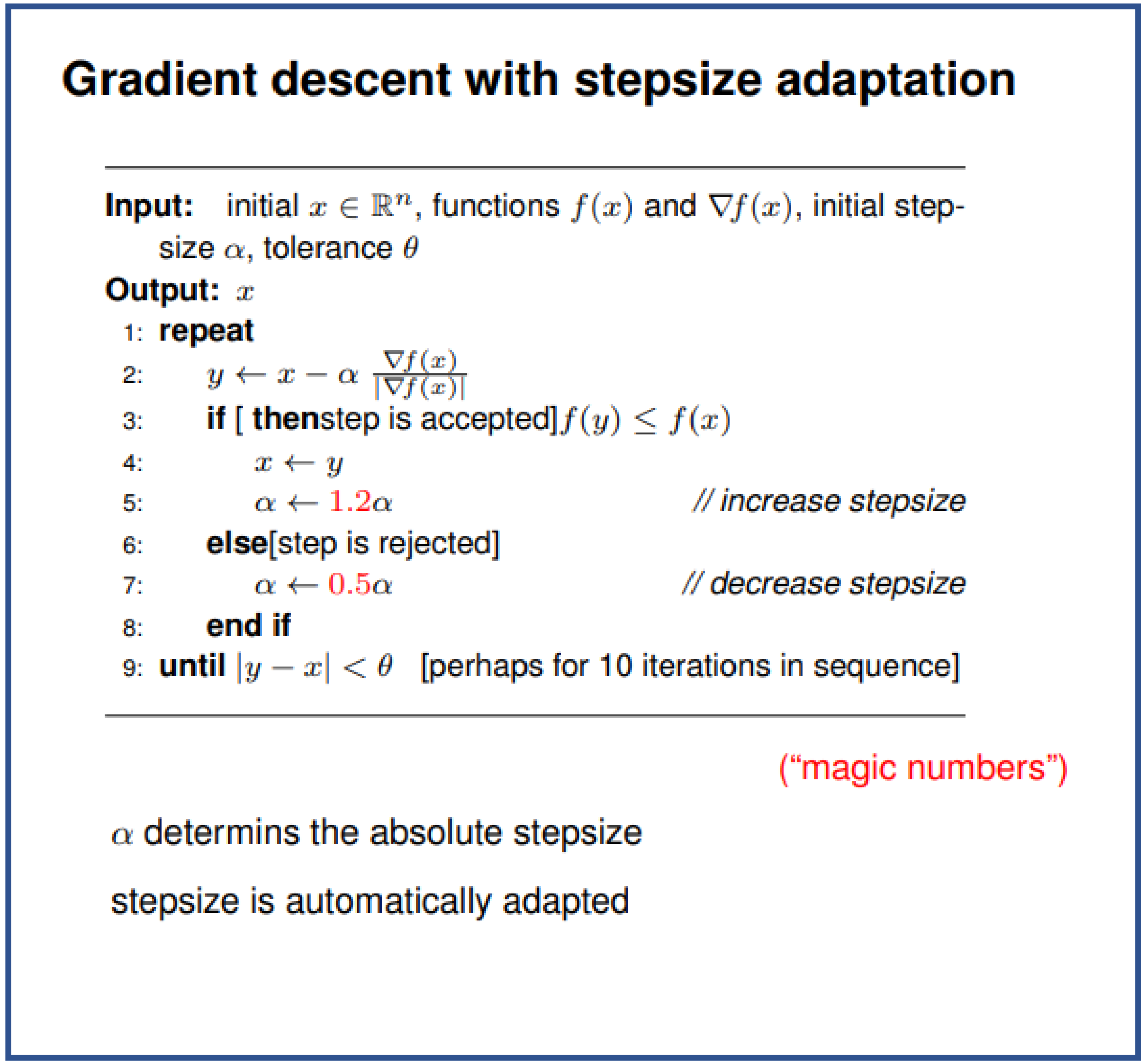
First order and second order optimization application

1st Order method & Pseudo algorithm

There are several methods:

- Plain gradient descent with adaptive step size
- Steepest descent
- Conjugate gradient
- Rprop (heuristic)

In this project submission, using plain gradient descent as example



Gradient descent with stepsize adaptation

```
Input: initial $x \in \mathbb{R}^n$, functions $f(x)$ and $\nabla f(x)$, initial step-
    size $\alpha$, tolerance $\theta$
Output: $x$
1: repeat
2:     $y \leftarrow x - \alpha \frac{\nabla f(x)}{|\nabla f(x)|}$
3:     if [ thenstep is accepted]$f(y) \le f(x)$
4:         $x \leftarrow y$
5:         $\alpha \leftarrow 1.2\alpha$                // increase stepsize
6:     else[step is rejected]
7:         $\alpha \leftarrow 0.5\alpha$                // decrease stepsize
8:     end if
9: until $|y - x| < \theta$  [perhaps for 10 iterations in sequence]
```

("magic numbers")

α determins the absolute stepsize

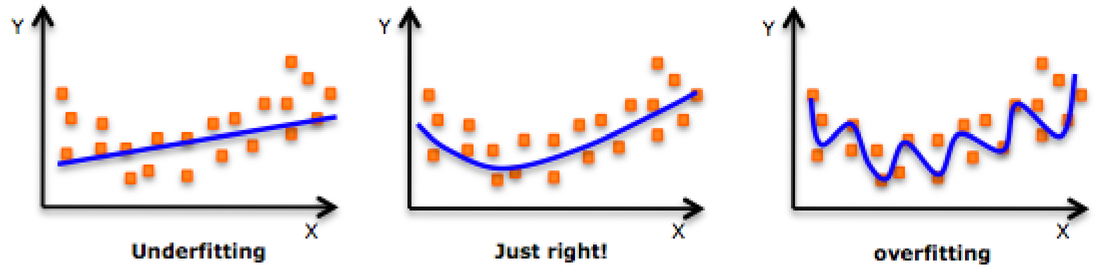
stepsize is automatically adapted

First order and second order optimization application

1st Order method and Overfitting

In designing optimization algorithm, we do pursue a goodness of fit so that a set of observations are fitted

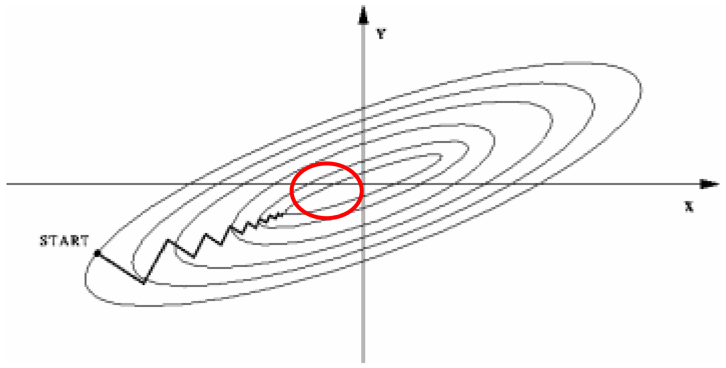
But simple 1st order method can not achieve when having following challenges



Challenges with "simple" 1st order method for "over-fitting":

- Highly sensitive to ill-conditioning
- Very difficult to tune (many) hyper-parameters

- Avoids Over-fitting?



"Over-fitting" is difficult with "simple" 1st order method!

First order and second order optimization application

There are several 2nd order methods:

- Newton
- Gauss-newton
- Quasi-Newton

Remedy?

1. "Not-So-Simple" 1st order method, e.g., accelerated *and* adaptive

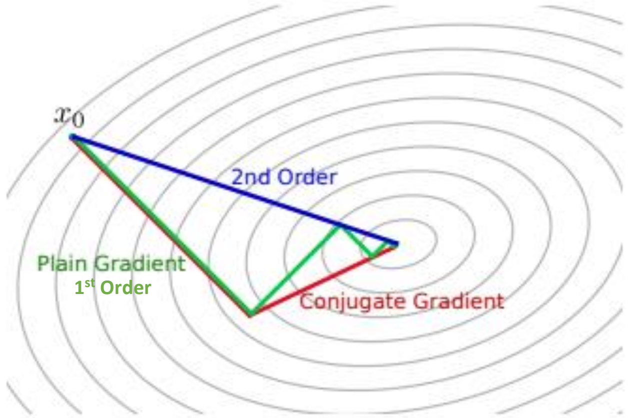
2. 2nd order methods, e.g., methods

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - [\nabla^2 F(\mathbf{x}^{(k)})]^{-1} \nabla F(\mathbf{x}^{(k)})$$

First order and second order optimization application

2nd order algorithm advantage

- Better direction:



- Better stepsize:
  - a *full step* jumps directly to the minimum of the local squared approx.
  - often this is already a good heuristic
  - additional stepsize reduction and dampening are straight-forward

First order and second order optimization application

Before 2nd order optimization methods,
Let's review the constrained optimization (Lagrangian we already familiar with)

- General constrained optimization problem:

  Let $x \in \mathbb{R}^n$, $f: \mathbb{R}^n \to \mathbb{R}$, $g: \mathbb{R}^n \to \mathbb{R}^m$, $h: \mathbb{R}^n \to \mathbb{R}^l$ find

$$\min_x \; f(x) \quad \text{s.t.} \quad g(x) \le 0, \; h(x) = 0$$

The Lagrangian

- Given a constraint problem

$$\min_x \; f(x) \quad \text{s.t.} \quad g(x) \le 0$$

  we define the **Lagrangian** as

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

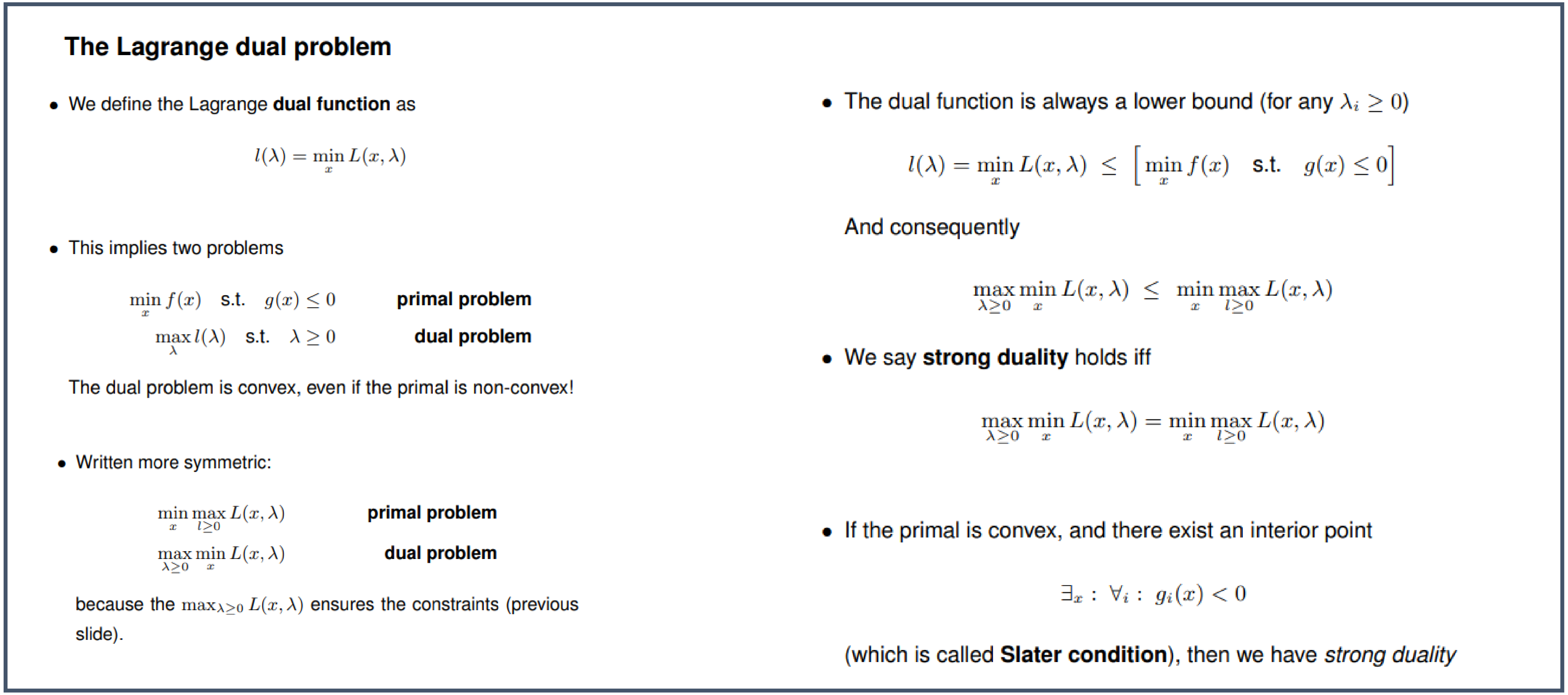
- The $\lambda_i \ge 0$ are called **dual variables** or *Lagrange multipliers*

What's the point of this definition?

- The Lagrangian is useful to compute optima analytically, on paper – that's why physicist learn it early on
- The Lagrangian implies the KKT conditions of optimality
- Optima are necessarily at saddle points of the Lagrangian
- The Lagrangian implies a dual problem, which is sometimes easier to solve than the primal

First order and second order optimization application

Before deep dive into 2nd order optimization methods
Let's recap constrained optimization (Lagrangian we are familiar with)



The Lagrange dual problem

- We define the Lagrange **dual function** as

$$l(\lambda) = \min_x L(x, \lambda)$$

- This implies two problems

$$\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0 \qquad \textbf{primal problem}$$
$$\max_\lambda l(\lambda) \quad \text{s.t.} \quad \lambda \geq 0 \qquad \textbf{dual problem}$$

The dual problem is convex, even if the primal is non-convex!

- Written more symmetric:

$$\min_x \max_{l \geq 0} L(x, \lambda) \qquad \textbf{primal problem}$$
$$\max_{\lambda \geq 0} \min_x L(x, \lambda) \qquad \textbf{dual problem}$$

because the $\max_{\lambda \geq 0} L(x, \lambda)$ ensures the constraints (previous slide).

- The dual function is always a lower bound (for any $\lambda_i \geq 0$)

$$l(\lambda) = \min_x L(x, \lambda) \leq \left[\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0 \right]$$

And consequently

$$\max_{\lambda \geq 0} \min_x L(x, \lambda) \leq \min_x \max_{l \geq 0} L(x, \lambda)$$

- We say **strong duality** holds iff

$$\max_{\lambda \geq 0} \min_x L(x, \lambda) = \min_x \max_{l \geq 0} L(x, \lambda)$$

- If the primal is convex, and there exist an interior point

$$\exists_x : \forall_i : g_i(x) < 0$$

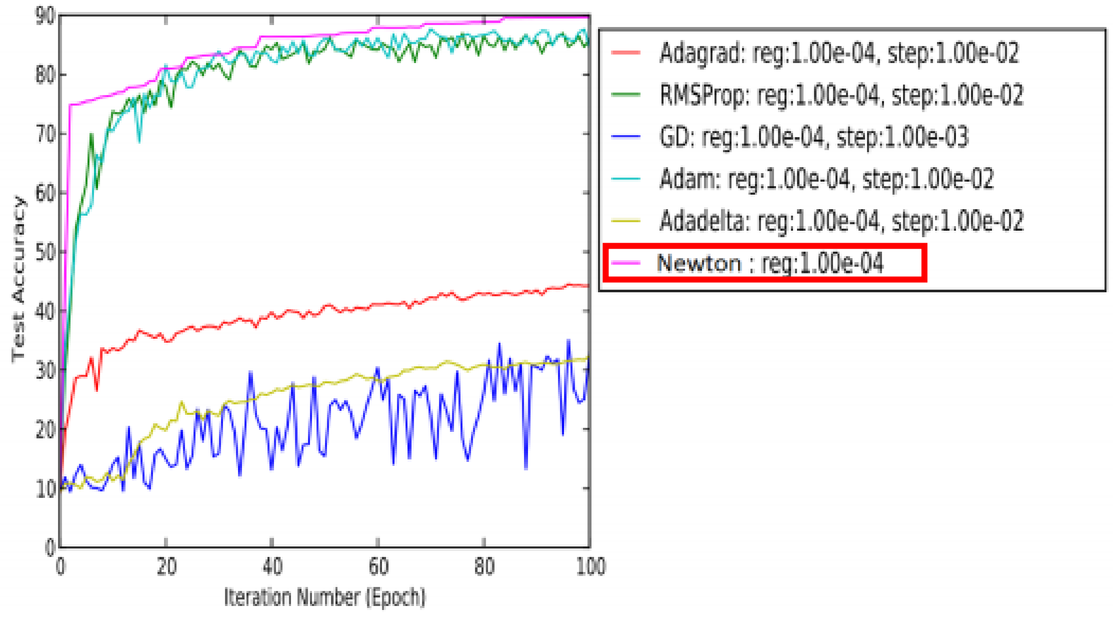
(which is called **Slater condition**), then we have *strong duality*

First order and second order optimization application

2nd order optimization methods features:

- Use both gradient and Hessian information
- Fast convergence rate
- Resilient to ill-conditioning
- They "over-fit" nicely!
- However, per-iteration cost is high!

$n : 50,000, p = 528$, No. Classes $= 11, \lambda : 0.0001$



First order and second order optimization application

2nd order optimization methods features:

- *Notation:*

  objective function: $f : \mathbb{R}^n \to \mathbb{R}$

  gradient vector: $\nabla f(x) = \left[\frac{\partial}{\partial_x} f(x)\right]^\top \in \mathbb{R}^n$

  Hessian (symmetric matrix):

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2}{\partial_{x_1}\partial_{x_1}} f(x) & \frac{\partial^2}{\partial_{x_1}\partial_{x_2}} f(x) & \cdots & \frac{\partial^2}{\partial_{x_1}\partial_{x_n}} f(x) \\ \frac{\partial^2}{\partial_{x_1}\partial_{x_2}} f(x) & & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial^2}{\partial_{x_n}\partial_{x_1}} f(x) & \cdots & \cdots & \frac{\partial^2}{\partial_{x_n}\partial_{x_n}} f(x) \end{pmatrix} \in \mathbb{R}^{n\times n}$$

  Taylor expansion:

$$f(x') = f(x) + (x' - x)^\top \nabla f(x) + \frac{1}{2}(x' - x)^\top \nabla^2 f(x)\,(x' - x)$$

- *Problem:*

$$\min_x f(x)$$

  where we can evaluate $f(x)$, $\nabla f(x)$ and $\nabla^2 f(x)$ for any $x \in \mathbb{R}^n$

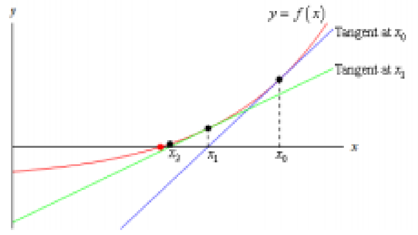
First order and second order optimization application

2nd order optimization methods

- Newton method & Pseudo algorithm

Newton method

- For finding roots (zero points) of $f(x)$



$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

- For finding optima of $f(x)$ in 1D:

$$x \leftarrow x - \frac{f'(x)}{f''(x)}$$

For $x \in \mathbb{R}^n$:

$$x \leftarrow x - \nabla^2 f(x)^{-1} \nabla f(x)$$

- Often, $\nabla^2 f(x)$ is banded (non-zero around diagonal only)

  $\rightarrow Ax = b$ becomes super fast using `dpbsv` (Dynamic Programming)

Newton method with adaptive stepsize α

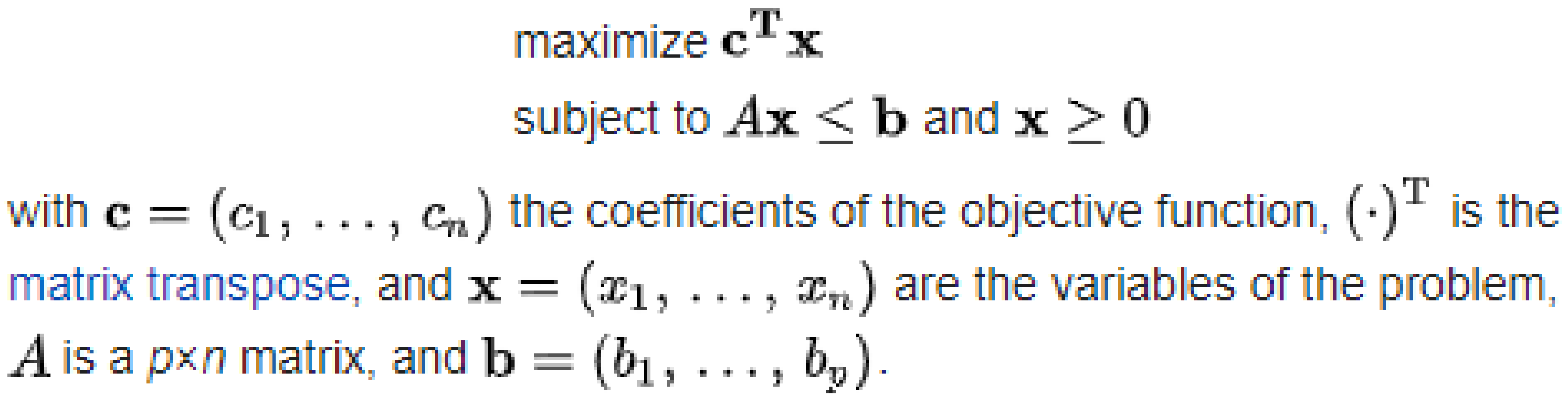
Input: initial $x \in \mathbb{R}^n$, functions $f(x), \nabla f(x), \nabla^2 f(x)$, tolerance θ
Output: x

```
1: initialize stepsize $\alpha = 1$ and damping $\lambda = 10^{-10}$
2: repeat
3:     compute $\Delta$ to solve $(\nabla^2 f(x) + \lambda \mathbf{I})\, \Delta = -\nabla f(x)$
4:     repeat                                          // "line search"
5:         $y \leftarrow x + \alpha\Delta$
6:         if $f(y) \le f(x)$ then                     // step is accepted
7:             $x \leftarrow y$
8:             $\alpha \leftarrow \alpha^{0.5}$        // increase stepsize towards $\alpha = 1$
9:         else                                        // step is rejected
10:            $\alpha \leftarrow 0.1\alpha$           // decrease stepsize
11:        end if
12:    until step accepted or (in bad case) $\alpha\|\Delta\|_\infty < \theta/1000$
13: until $\|\Delta\|_\infty < \theta$
```

- Notes:
  - Line 3 computes the Newton step $\Delta = \nabla^2 f(x)^{-1} \nabla f(x)$,
    use special Lapack routine `dposv` to solve $Ax = b$ (using Cholesky decomposition)
  - λ is called **damping**, makes the parabola more "steep" around current x
    for $\lambda \rightarrow \infty$: Δ becomes colinear with $-\nabla f(x)$ but $|\Delta| = 0$

Case study: Classic Portfolio Optimization

We are interested to see how linear programming for first order optimization applications in industry are utilized and mathematically formulated.
Firstly, let me introduce Simplex, it operates on linear programs in the following form:



maximize $\mathbf{c}^{\mathrm{T}}\mathbf{x}$

subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$

with $\mathbf{c} = (c_1, \ldots, c_n)$ the coefficients of the objective function, $(\cdot)^{\mathrm{T}}$ is the matrix transpose, and $\mathbf{x} = (x_1, \ldots, x_n)$ are the variables of the problem, A is a *p*×*n* matrix, and $\mathbf{b} = (b_1, \ldots, b_p)$.

Classic Portfolio Optimization

Given Total carrying cash: 1 million dollars on hand

Then we decide to hold stock portfolio with m stocks to choose and allocate to portfolio

Objective: Max(return) and Min(risk), where min(risk) = Max(-risk)

<u>Decision variables:</u>

w1, w2, ….w_m: w is the percentage of cash to be allocated to m options of stocks, w has domain in range of 0 to 1, summation of w = 100%

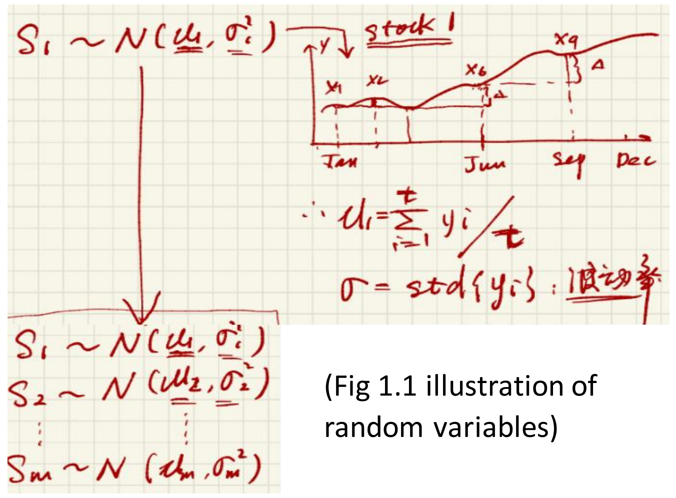
<u>Objective function:</u>

Minimize (-return + lambda*risk), where lambda: a control variable which maintaining weighted average of return and risk, i.e. lambda increasing(decreasing) ➔ emphasis on return(risk)

Classic Portfolio Optimization

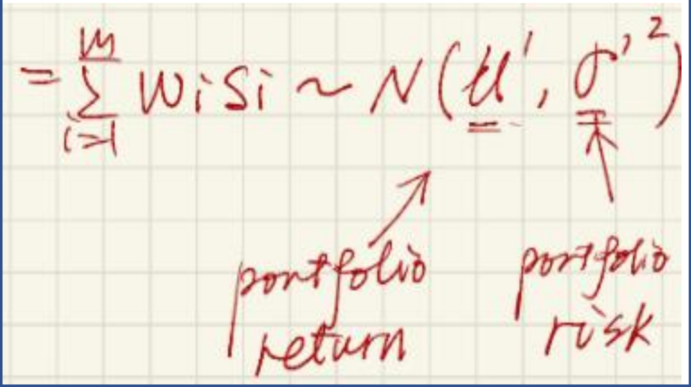
We incorporate the Random variables: Si, Si is assumed to follow normal distribution ~N(average return, risk_i^2), To be more specific, here is the examples for Si:

- S1 ~ N(U_1, sigma_1^2)
- U_1: 1/t*summation(y1) in period of t
- sigma_1 = standard deviation (y1) in period of t


$S_1 \sim N(u_1, \sigma_i^2)$
stock 1
y
x1
x2
x6
x9
Δ
Jan
Jun
Sep
Dec
$\therefore u_1 = \sum_{i=1}^{t} y_i / t$
$\sigma = std\{y_i\}$: 波动率
$S_1 \sim N(u_1, \sigma_i^2)$
$S_2 \sim N(u_2, \sigma_2^2)$
$S_m \sim N(u_m, \sigma_m^2)$

(Fig 1.1 illustration of random variables)

- Therefore, the summation of applied weighted stock times stock distribution also follows Gaussian Distribution:



Classic Portfolio Optimization

- Where the gaussian distribution for both return and risk can be formatted as:

$$\Rightarrow \quad \mu = \sum_{i=1}^{m} w_i \mu_i$$

$$\sigma^2 = \sum_{i=1}^{m} w_i w_j \sigma_{ij} \; (\mathrm{cov})$$

- Now we can reformat the Objective Function as:

$$\text{minimize} \quad -\sum_{i=1}^{m} w_i \mu_i + \lambda \sum_{i=1}^{m} w_i w_j \sigma_{ij}$$

Classic Portfolio Optimization

- The algebra matrix per each variable are as following:

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_m \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_1^2, \sigma_1\sigma_2 & \cdots & \sigma_1\sigma_m \\ & & \vdots \\ \sigma_m\sigma_1 & \cdots & \sigma_m^2 \end{bmatrix}$$

- Then we successfully transformed our original objective function to be:

$$\therefore \text{ minimize } -\mu^T w + \lambda w^T \Sigma w$$

$$\text{s.t. } \sum_{i=1}^{m} w_i = 1, \quad w_i \geq 0$$

Classic Portfolio Optimization

Conclusion:

Lastly, our goal is just to prove this problem is convex program so that we can apply linear programming to find the optimal, this is apparent that the domain of variables set and convex set, and the objective function we defined is quadratic, so it is also convex.

Therefore, based on above justification, we can include that this portfolio program can be solved using linear programming as it follows all convex optimization principles.