Iordanis Koutsopoulos

# NETWORK OPTIMIZATION LECTURE NOTES

## Part 2 [1]

[1] Based on Notes from Fall 2006, and Fall 2010. Organized by Stella Gaki.

# Contents

# 1 Example: Optimization problem formulation

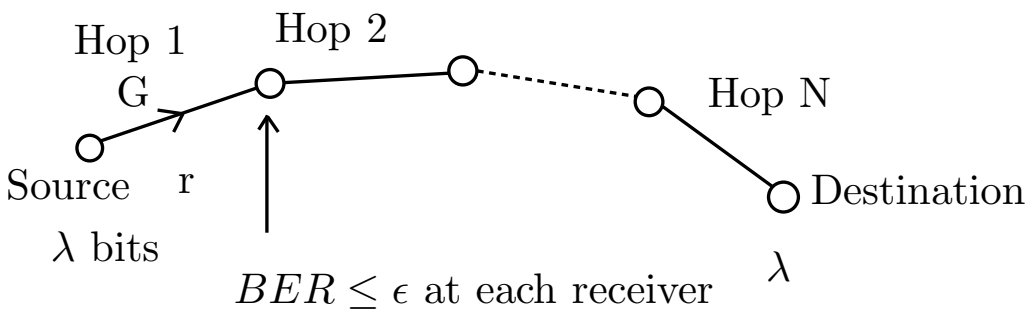


Fig.3 : Multi-hop transmission in a sensor network

Figure 1: Bit transfer from a source to a destination in multi-hop in a wireless sensor network.

Consider a sensor network, in which $\lambda$ bits have to be transmitted from a source (S) to a destination (D). Destination can be a processing (fusion) center.

The $\lambda$ bits are generated by the sensing / measurement process at the sensor in (S). The bits need to be forwarded through the multi-hop network from S to D. At each hop, $\lambda$ bits are transmitted from the transmitter to the corresponding receiver. Here, the multi-hop path to the destination is *known*. Suppose the number of hops is $N$.

There exist $L$ possible data transmission rates, $r_1, ...r_L$.

Each transmitter $i = 1, \ldots, N-1$ has to choose a rate $r_i \in \{r_1, \ldots, r_L\}$ out of the rate set. Each rate $r_i$ denotes bits/symbol that can be transmitted and can be mapped to choosing the *level of modulation* transmitter $i$.

For example, for QPSK and 8-QAM modulation, $r = 2$ and $r = 3$ bits/symbol respectively.

**Reminder from Digital Comm:** To transmit data, a transmitter divides the stream of coming bits into teams of $r$ bits. E.g. for BPSK, QPSK, 8-QAM modulation, $r = 1, 2, 3$ respectively. The transmitter then "loads" the $r$ in a pulse (symbol), so that we have that it transmits $r$ bits/symbol. The transmitter circuitry generates pulses with rate $s$ symbols/sec (symbol rate).

Fix attention at hop $i$. Suppose $t_i$ is the time duration of transmission of the $\lambda$ bits. Then,

$$\lambda(bits) = \frac{\text{bits}}{\text{symbol}} \cdot \frac{\text{symbols}}{\text{sec}} \cdot \text{sec} \tag{1}$$

$$\Rightarrow \lambda = r_i \cdot s \cdot t_i \tag{2}$$

where $s$ is the symbol rate in symbols/sec, with $s = \frac{1}{T_s}$ , where $T_s$ is the symbol (pulse) duration. Bits are loaded on symbol pulses (which can be square pulses or sigmoid ones).

Rate $r_i$ determines the number of transmitted bits per symbol in hop $i$, namely it determines the *modulation level*. The total time needed for the transmission of the $\lambda$ bits across hops,

is:

$$\sum_{i=1}^{N} t_i = \frac{\lambda}{s} \sum_{i=1}^{N} \frac{1}{r_i} \, . \tag{3}$$

In this problem, the modulation level $r_i$ for the transmitter of each hop $i$ is controllable. In fact, we assume that $r_i$ is continuous variable.

At each receiver along the path, the SNR should satisfy:

$$\text{SNR}_i \geq -\frac{\ln(5\epsilon)}{1.5}(2^{r_i} - 1) \tag{4}$$

if rate $r_i$ is used at transmitter $i$, so that BER $\leq \epsilon$ at each receiver, where $\epsilon$ is a prespecified number (e.g.$10^{-6}, 10^{-7}$) that shows the maximum tolerable bit error rate (BER) at each transmission hop. The inequality above results from the following approximate formula for BER:

$$\text{BER} \approx \frac{1}{5} e^{-\frac{1,5 \cdot SNR}{2^r - 1}} \, . \tag{5}$$

The relation $\text{SNR}_i \, (2^{r_i} - 1)$ may also be derived from the Shannon capacity formula, $r_i = \log_2(1 + \text{SNR}_i)$.

The receiver is aware of transmitter rate and tries to decode the signal according to that. If $r_i$ is too large, this means that signal points in the constellation diagram are too close to each other, so there is a difficulty in distinguishing what has been sent. Thus, good quality channel is needed to reduce the errors.

The minimum required SNR is related to transmission power as follows:

$$\text{SNR}_i = \frac{g_i P_i}{\sigma^2} \tag{6}$$

where $g_i \in [0, 1]$ is the link gain between transmitter-receiver at hop $i$ (it shows the losses in the power of the transmitted signal) and $\sigma^2$ is the noise power at receiver. By using equality and solving for $P_i$ we have the following:

$$P_i = \frac{\sigma^2 a}{g_i} \left(2^{r_i} - 1\right) \approx k \cdot 2^{r_i} \tag{7}$$

where $a = -\frac{\ln(5\epsilon)}{1.5}$ and $k$ is a proportionality constant. Thus $P_i$ is exponential in rate $r_i$. The energy consumed for transmission at hop $i$ is:

$$\text{Energy}(joules) = \text{Power}(Watt) \times \text{time}(sec) \tag{8}$$

¿From equation (2) the energy is :

$$E_i = P_i \cdot \frac{\lambda}{s} \cdot r_i \tag{9}$$

$$\Rightarrow \qquad E_i = \frac{k \cdot 2^{r_i}}{\frac{s \cdot r_i}{\lambda}}$$

$$\Rightarrow \qquad E_i = k \cdot 2^{r_i} \cdot \frac{\lambda}{s r_i}$$

*What is the optimization problem that arises ?*

(a) Given a route from S to D, what is the transmission rate $r_i$ in each hop $i$ so that the total energy $\sum_{i=1}^{N} E_i$ is minimized?

If there are no constraints on the time by which the $\lambda$ bits need to be transferred, and since

$$E_i = k_i \cdot \frac{2^{r_i}}{r_i} \tag{10}$$

and there are $L$ choices for the rate, $r \in \{b_1, ldots, b_L\}$, with $b_1 < b_2 < \ldots < b_L$, the optimal choice is to operate with the minimum rate for each hop, namely $\forall$ hops $i$, choose $r_i = b_1$.

(b) *What happens when there is a deadline by which bits need to be transferred to the destination?*

Deadlines arise since the information that needs to be transmitted is time-critical. Then $\lambda$ bits need to be transferred from source to destination within some deadline time $D$. We also assume that only one hop is active at a time, the one that transmits.

There are two conflicting views: Sum $\sum_{i=1}^{N} E_i$ needs to be low (because in a sensor network battery consumption has to be low). That means that energy transmission in each hop has to be low. Thus, in order to have low energy consumption, each $r_i$ has to be low. But, on the other hand, if each $r_i$ is low, it takes longer for the $\lambda$ bits to be transmitted and the deadline $D$ is more difficult to respect. Therefore, we can formulate the following optimization problem (assuming that the rates are

continuous variables):

$$\min_{\mathbf{r}} \sum_{i=1}^{N} k_i \cdot \frac{2^{r_i}}{r_i},$$

$$\text{subject to: } \frac{\lambda}{s} \sum_{i=1}^{N} \frac{1}{r_i} \leq D$$

with $\mathbf{r} = \{r_1, ...r_N\}$, $\mathbf{r} \geq \mathbf{0}$.

Objective function $f(\mathbf{r}) = \sum_{i=1}^{N} k_i \frac{2^{r_i}}{r_i}$ is convex in $\mathbf{r}$ , since it is the sum of convex functions of the form $\frac{e^x}{x}$. Also the constraints are convex. Therefore, we have a convex optimization problem.

**Remark 1:** Another optimization problem could be formed, where the total delay in transferring the information would be minimized, subject to a constraint on the total energy consumption.

**Remark 2:** In the formulation above, we assumed that the link gains $g_i$ for each hop $i$ are fixed quantities. A large $g_i$ (good channel, few losses) means that smaller energy is needed to achieve a certain BER at the corresponding receiver. Gain $g_i$ can be measured as follows. The transmitter sends a signal of known power $p$ (known, also to the receiver). The receiver then measures the received signal power, $p'$ and finds gain $g = p'/p$.

**Remark 3:** The optimization problem formulated above assumes that all gains $g_1, \ldots, g_{N-1}$ are known to a central controller node which then computes the solution and sends it to each transmitter. However, another version of the problem, *an online one* can be as follows. Each transmitter $i$ only knows its own gain $g_i$ and decides (perhaps greedily) on its own transmit rate $r_i$, based on that and the remaining time until the deadline. Then the next transmitter $i+1$ get the bits and decides on its own rate $r_{i+1}$ and so on. In this second version of the problem, not all the information is available a priori, but it is revealed gradually at each node.

# 2 Descent Methods

We want to minimize a function of many variables, $f(\mathbf{x})$. There are no constraints to this problem.

A first way to solve the problem is to apply the sufficient conditions that involve $\nabla f(\mathbf{x})$ and the Hessian matrix. However, this requires the gathering of all variables to a central point, and the solution of (possibly complicated) equations. Alternatively, we can use the Descent methods.

*Descent methods* are iterative numerical methods which, starting with an initial guess $\mathbf{x}^{(0)}$, improve gradually the value of the objective function $f$. To do so, they move from one point

to the next one, by applying the general rule:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}\,, \tag{11}$$

where $\alpha$ is a positive scalar parameter called the *step size* at iteration $k$, and $\mathbf{d}$ is a direction vector, $\mathbf{d} \in \mathcal{R}^n$, called the *descent* direction of iteration $k$.

Thus, the descent method generates a *sequence* of points $\mathbf{x}^{(0)} \rightarrow \mathbf{x}^{(1)} \rightarrow \ldots \rightarrow \mathbf{x}^{*}$. The sequence $\{\mathbf{x}^{(k)}\}$ converges to the point $\mathbf{x}^{*}$.

The point $\mathbf{x}^{*}$ is (at least) a local minimum of $f$. We symbolize the fact that $\mathbf{x}^{*}$ minimizes the value of $f$ as $\mathbf{x}^{*} = \arg\min_{\mathbf{x}} f(\mathbf{x})$.

At each iteration, it must be:

$$f(\mathbf{x}^{(k+1)}) < f(\mathbf{x}^{(k)}),\ \forall k\,. \tag{12}$$

## 2.1 Properties of search direction $\mathbf{d}$

How to choose the descent direction $\mathbf{d}^{(k)}$?

In a *descent method*, the descent direction must be such that $\nabla f(\mathbf{x}^{(k)})^T\mathbf{d}^{(k)} < 0$ for all $k$, so that the value of the function decreases with iteration $k$. In that case, the direction is called *descent direction* for $f$ at $\mathbf{x}^{(k)}$. That is, the search direction should make an *acute* angle with the direction of the gradient at $\mathbf{x}^{(k)}$ (otherwise it will be $f(\mathbf{x}^{(k+1)}) > f(\mathbf{x}^{(k)})$).

Indeed, since we want $f(\mathbf{x}^{(k+1)}) < f(\mathbf{x}^{(k)})$, and it is $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$, we recall the Taylor first order expansion formula to have: $f(\mathbf{x}^{(k+1)}) = f(\mathbf{x}^{(k)}) + \alpha^{(k)}\nabla f(\mathbf{x}^{(k)})^T\mathbf{d}^{(k)}$ and it must be $\nabla f(\mathbf{x}^{(k)})^T\mathbf{d}^{(k)} < 0$.

In the next figure below we depict some descent directions.

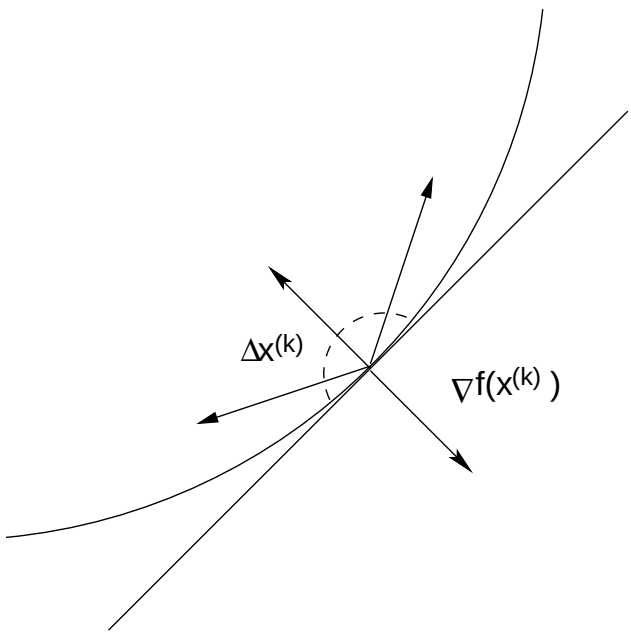


Figure 2: Descent directions for the Gradient Descent method.

# 3 Gradient Descent Method

The Gradient Descent Method is a special descent method where the search direction is: $\mathbf{d}^{(k)} = -\nabla f(\mathbf{x}^{(k)})$.

The iteration for the gradient descent method is:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha^{(k)} \nabla f(\mathbf{x}^{(k)}). \tag{13}$$

We will show that $f(\mathbf{x}^{(k+1)}) < f(\mathbf{x}^{(k)})$.

We have:

$$f(\mathbf{x}^{(k+1)}) = f(\mathbf{x}^{(k)}) - \alpha^{(k)} \nabla f(\mathbf{x}^{(k)}) \tag{14}$$

Using Taylor's expansion formula, equation becomes:

$$f(\mathbf{x}^{(k+1)}) = f(\mathbf{x}^{(k)}) - \alpha^{(k)} \|\nabla f(\mathbf{x}^{(k)})\|^2 \tag{15}$$

Since $\alpha^{(k)}$ and $\left\|\nabla f(\mathbf{x}^{(k)})\right\|^2$ are positive, we have $f(\mathbf{x}^{(k+1)}) < f(\mathbf{x}^{(k)})$.

**Note:** We will present various ways of how to determine $\alpha^{(k)}$ later.

**Remark:** Gradient methods can be efficient in systems / networks in which each node can control one component (variable) of the problem (i.e. of the vector $\mathbf{x}^{(k)}$). Then each node $i$ can run the gradient descent algorithm locally on its variable $x_i$ and thus get to optimal solution $x_i^*$. In these cases, the gradient descent method facilitates distributed computations.

## 3.1 Properties of Gradient Descent

1. The direction of $\nabla f(\mathbf{x}_0)$ is orthogonal to the tangent vector to an arbitrary smooth curve passing through $\mathbf{x}_0$ at the level set $f(\mathbf{x}) = f(\mathbf{x}_0)$ (see Figure 3).

**Remark:** The level set of a function $f$ at level $c \in \mathcal{R}$ is the set of points $\mathcal{S} = \{\mathbf{x} : f(\mathbf{x}) = c\}$.

The level set can be visualized as follows. Imagine a (say three-dimensional) function $f$ (imagine in the shape of a bell for example). Imagine we cut the bell with a horizontal plane at the vertical axis position $c$. What will remain is a circle. The set of points on the circle is the level set at level $c$.

2. $\nabla f(\mathbf{x})$ is always the direction of maximum rate of increase of $f$ at point $\mathbf{x}$.

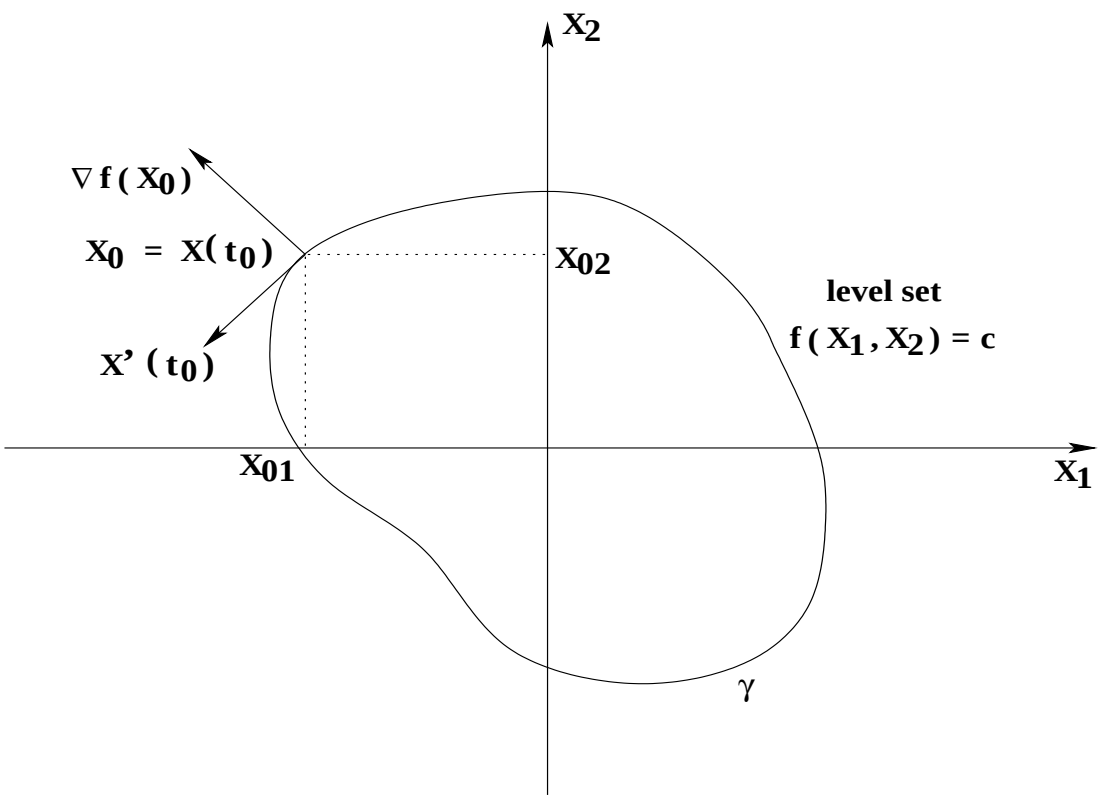


Figure 3: Orthogonality of the gradient to the level set

First, we will define the *directional derivative* of $f$ at direction $\mathbf{d}$.

**Directional derivative:** Define as $\frac{\partial f(\mathbf{x})}{\partial \mathbf{d}}$ the directional derivative of function $f : \mathcal{R}^n \longrightarrow \mathcal{R}$ in the direction $\mathbf{d}$ at point $\mathbf{x}$ as:

$$\frac{\partial f}{\partial \mathbf{d}} = \lim_{\alpha \to 0} \frac{f(\mathbf{x} + \alpha\, \mathbf{d}) - f(\mathbf{x})}{\alpha} = \mathbf{d}^T \nabla f(\mathbf{x}). \tag{16}$$

where $\mathbf{d}^T \nabla f(\mathbf{x})$ denotes the *rate of increase* of $f$ at direction $\mathbf{d}$ at point $\mathbf{x}$ and $\alpha$ is a small number.

When the direction vector becomes the gradient, i.e. $\mathbf{d} = \frac{\nabla f(\mathbf{x})}{||\nabla f(\mathbf{x})||}$, the above rate becomes maximum.

This can be proved if we apply the Cauchy-Schwartz inequality. This says that, for vectors $\mathbf{a}$ and $\mathbf{b}$, it is: $(\mathbf{a}^T\mathbf{b})^2 \leq ||\mathbf{a}||^2||\mathbf{b}||^2 \Rightarrow \mathbf{a}^T\mathbf{b} \leq ||\mathbf{a}||||\mathbf{b}||$ ), with equality if $\mathbf{a}, \mathbf{b}$ are co-linear.

Then,we have that:

$$\mathbf{d}^T \nabla f(\mathbf{x}) \leq ||\mathbf{d}||\, ||\nabla f(\mathbf{x})||\,. \tag{17}$$

Thus, the maximum value of the left side of expression (19) is $||\nabla f(\mathbf{x})||$ which is achieved for direction

$$\mathbf{d} = \frac{\nabla f(\mathbf{x})}{||\nabla f(\mathbf{x})||}, \tag{18}$$

namely the unit-norm direction of gradient.

Thus: the direction of gradient is the one along which the value of $f$ increases with maximal rate.

3. For each iteration $k$, the points produced by the iteration are such that:

$$\left(\mathbf{x}^{(k+2)} - \mathbf{x}^{(k+1)}\right) \perp \left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right)$$

Thus, vector $\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right)$ is orthogonal to vector $\left(\mathbf{x}^{(k+2)} - \mathbf{x}^{(k+1)}\right)$, which equivalently means: $\nabla f\left(\mathbf{x}^{(k+1)}\right) \perp \nabla f\left(\mathbf{x}^{(k)}\right)$, or $\nabla f\left(\mathbf{x}^{(k)}\right)$ is parallel to the tangent plane to the level set $\left\{f(\mathbf{x}) = f\left(\mathbf{x}^{(k+1)}\right)\right\}$ at $\mathbf{x}^{(k+1)}$.

A depiction of the possible iteration of the gradient method is given in the figure below.

**Remark 1:** For minimization problems, the gradient algorithm moves towards direction opposite to the one of gradient. For maximization problems, the algorithm moves towards the direction of the gradient. This direction is then called an ascent direction.

**Remark 2:** The Euclidean Norm of vector $\mathbf{d}$ indicates the velocity with which $f$ changes in every point (i.e. the magnitude of $\mathbf{d}$ indicates the greatest rate of change).

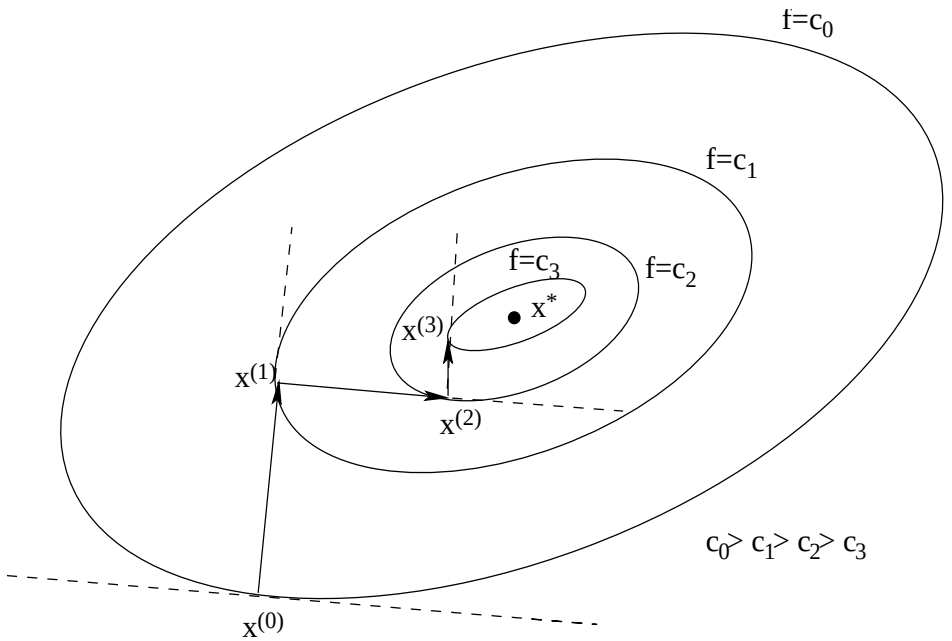


Figure 4: Point Sequence in the gradient descent method.

## 3.2 General descent method algorithm

**The algorithmic steps for a General Descent Method**

Start with initial point $\mathbf{x}^{(0)}$.

At each iteration $k$:

1. Determine a descent direction $\mathbf{d}^{(k)}$.

2. Determine a step $\alpha^{(k)}$.

3. Update the point according to equation:

$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$

*STOP* if $\left\|\nabla f\left(\mathbf{x}^{(k)}\right)\right\| \leq \varepsilon$, where $\varepsilon$ a small positive number.

Another (more general) form of descent methods is as follows:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha^{(k)} D_k \nabla f\left(\mathbf{x}^{(k)}\right) \tag{19}$$

where $D_k$ is a symmetric, positive definite matrix. The descent direction is $\mathbf{d}^{(k)} = -D_k \cdot \nabla f(\mathbf{x}^{(k)})$. It is $\nabla f(\mathbf{x}^{(k)})^T \mathbf{d}^{(k)} < 0$ as it should be for all descent methods, since $D_k$ is positive definite.

Note that this more general form can be used to weigh appropriately each component of Gradient vector.

**Note** that for $D_k = I$ (where $I$ is the unit matrix), we get the Gradient's descent method iteration and for $D_k = [\nabla^2 f(\mathbf{x}^{(k)})]^{-1}$, we get the Newton's method iteration.

## 3.3 Gradient method with Constrains on $\mathbf{x}$

The gradient methods above work when $\mathbf{x} \in \mathcal{R}^n$. In many problems however, especially in networks, the variables of vector $\mathbf{x}$ are subject to various constraints. A most common constraint is the fact that these are non-negative (i.e. $\mathbf{x} \geq 0$ ). Then, for problem $\min_{\mathbf{x} \geq 0} f(\mathbf{x})$, the iteration for the Gradient descent method, for each component $i$ is:

$$x_i^{(k+1)} = (x_i^{(k)} - \alpha^{(k)} \nabla \frac{\partial f(\mathbf{x}^{(k)})}{\partial x_i})^+ \tag{20}$$

where $\mathbf{x}^+$ is equal to $\mathbf{x}$ if $x > 0$, and zero for negative values of $\mathbf{x}$.

# 4 Upper and lower bounds on Hessian matrix of $f$

Suppose that $f$ is defined in a domain $\Omega$ and is a strictly convex function. Then there exist real numbers $m, M > 0$ such that:

$$m\,I \leq \nabla^2 f(\mathbf{x}) \leq M I \tag{21}$$

where $0 < m \leq M$ and $I$ is the identity matrix of size $n$ (assuming $\mathbf{x}\epsilon\mathcal{R}^n$).

**Proof of lower bound :** Since $f$ is convex, Hessian matrix is positive definite ($\nabla^2 f(\mathbf{x}) > 0$), that is, $y^T\,\nabla^2 f(\mathbf{x})\,y > 0$, $\forall y \neq 0$. Then $\exists m$ such that:

$$\begin{aligned} y^T\,(\nabla^2 f(\mathbf{x}) - m\,I)\,y > 0 \;\; \forall y \neq 0 \\ \Rightarrow \nabla^2 f(\mathbf{x}) \geq m\,I \end{aligned} \tag{22}$$

**Proof of upper bound :** Assume that $\lambda(\mathbf{x})$ is the eigenvalue of $\nabla^2 f(\mathbf{x})$ and $\boldsymbol{omega}(\mathbf{x})$ is the corresponding eigenvector of $\nabla^2 f(\mathbf{x})$. This means that:

$$\nabla^2 f(\mathbf{x})\,\boldsymbol{\omega}(\mathbf{x}) = \lambda(\mathbf{x})\,\mathbf{y}. \tag{23}$$

Note that $\lambda(\mathbf{x})$ and $\boldsymbol{\omega}(\mathbf{x})$ depend on the particular point $\mathbf{x}$, since $\nabla^2 f(\mathbf{x})$ depends on $\mathbf{x}$ and that $\boldsymbol{\omega}(\mathbf{x}) > 0$, since $\nabla^2 f(\mathbf{x})$ is positive definite.

Define $\Lambda$ as follows:

$$\Lambda = \max_{\mathbf{x} \in \Omega} \lambda(\mathbf{x}) \tag{24}$$

which leads us to inequality:

$$\nabla^2 f(\mathbf{x})\, \boldsymbol{\omega}(\mathbf{x}) \;\leq\; \Lambda\, \boldsymbol{\omega}(\mathbf{x}). \tag{25}$$

Multiply from the left by $\boldsymbol{\omega}(\mathbf{x}) > 0$ to get:

$$\boldsymbol{\omega}^T(\mathbf{x})\, \nabla^2 f(\mathbf{x})\, \boldsymbol{\omega}(\mathbf{x}) \;\leq\; \boldsymbol{\omega}^T(\mathbf{x})\, \Lambda\, \boldsymbol{\omega}(\mathbf{x}) = \boldsymbol{\omega}(\mathbf{x})^T\, \Lambda\, I\, \boldsymbol{\omega}(\mathbf{x}) \tag{26}$$

and finally set $M = \Lambda$ to get:

$$\nabla^2 f(\mathbf{x}) \;\leq\; MI. \tag{27}$$

## 4.1 Condition Number

Note that the ratio

$$\frac{\max_{\mathbf{x} \in \Omega} \lambda(\mathbf{x})}{\min_{\mathbf{x} \in \Omega} \lambda(\mathbf{x})} \geq \frac{M}{m} \tag{28}$$

is called the *Condition Number* of the Hessian matrix of function $f$.

Later, we will show that the condition number is related to the speed of convergence of gradient descent methods.

More information about these upper and lower bounds can be found in section 9.1.2 of the book ”Convex Optimization” by Boyd and Vandenberghe.

In the special case that $f(\mathbf{x})$ is quadratic, i.e.

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} \tag{29}$$

with $Q$ $n \times n$ positive definite matrix, it is $\nabla^2 f(\mathbf{x}) = Q$ and the condition number is defined as the ratio of its largest eigenvalue to its smallest eigenvalue.

$$\text{ConditionNumber} = \frac{\lambda_{\text{max}}(Q)}{\lambda_{\text{min}}(Q)} \tag{30}$$

where the maximum and minimum eigenvalues are now fixed numbers.

The condition number gives a measure of:

- The convergence of Gradient Method (i.e. how fast the sequence of points of the Gradient algorithm converges to a minimum $\mathbf{x}^*$).
- The anisotropy (eccentricity) of the set $\Omega$ in the case of generic functions $f(\mathbf{x})$.
- The distance of $f(\mathbf{x}^{(k)})$ from the optimal value of objective function (i.e. $f(\mathbf{x}^*)$) for every iteration $k$ (see the proof below).

**Proof**:

We will show that:

$$\frac{1}{2M}\|\nabla f(\mathbf{x})\|^2 \leq f(\mathbf{x}^{(k)}) - f(\mathbf{x}^*) \leq \frac{1}{2m}\|\nabla f(\mathbf{x})\|^2 \tag{31}$$

**Proof of upper bound:**

¿From Taylor's expansion formula we have:

$$f(\mathbf{y}) = f(\mathbf{x}) + \nabla^T f(\mathbf{x})\,(\mathbf{y}-\mathbf{x}) + \frac{1}{2}\,(\mathbf{y}-\mathbf{x})^T\,\nabla^2 f(\mathbf{x})\,(\mathbf{y}-\mathbf{x}) \tag{32}$$

Also from the bound above:

$$(\mathbf{y}-\mathbf{x})^T\,\nabla^2 f(\mathbf{x})\,(\mathbf{y}-\mathbf{x}) \;\geq m\;\|\mathbf{y}-\mathbf{x}\|^2 \tag{33}$$

¿From the above, and after bounding further the second term of the right hand side of the inequality, we have:

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \underbrace{\min_{\mathbf{y}}\left\{\nabla^T f(\mathbf{x})\,(\mathbf{y}-\mathbf{x}) \;+\; \frac{m}{2}\,\|\mathbf{y}-\mathbf{x}\|^2\right\}}_{\mathbf{g(y)}} \tag{34}$$

We can now find that minimum value of $g(\mathbf{y})$ with respect to $\mathbf{y}$, and this is $g(\mathbf{y}^*) = -\,\frac{1}{\mathbf{2m}}\;\|\nabla f(\mathbf{x})\|^2$, achieved for $\mathbf{y}^* = \mathbf{x} - \nabla f(\mathbf{x})/m$.

Therefore we have:

$$f(\mathbf{y}) \geq f(\mathbf{x}) \;-\; \frac{1}{\mathbf{2m}}\;\|\nabla f(\mathbf{x})\|^2 \tag{35}$$

Finally setting the $\mathbf{y} \rightarrow \mathbf{x}^*$ we take:

$$f(\mathbf{x}) \; - \; f(\mathbf{x}^*) \leq \frac{1}{2m} \left\| \nabla f(\mathbf{x}) \right\|^2 \tag{36}$$

In similar spirit, we can prove the **lower bound** starting with inequality:

$$(\mathbf{y} - \mathbf{x})^T \, \nabla^2 f(\mathbf{x}) \, (\mathbf{y} - \mathbf{x}) \; \leq M \; \left\| \mathbf{y} - \mathbf{x} \right\|^2 \; . \tag{37}$$

# 5 Convergence of Gradient Method

The gradient algorithm with fixed step size ($\alpha^{(k)} \; = \; \alpha$) converges to the minimum $\mathbf{x}^*$ in at most

$$k = \frac{\log(\frac{|f(\mathbf{x}^{(0)}) - \mathbf{x}^*|}{\varepsilon})}{\log \frac{1}{1 - \frac{m}{M}}} \tag{38}$$

$$\Rightarrow k \approx \frac{M}{m} \; \log(\frac{|f(\mathbf{x}^{(0)}) - \mathbf{x}^*|}{\varepsilon}) \tag{39}$$

steps. Here, $\varepsilon$ is a termination condition and satisfies the following relation: $\left| f(\mathbf{x}^{(k)}) - f(\mathbf{x}^*) \right| \leq \varepsilon$.

*The number of steps (and thus the delay) it takes for the algorithm to converge depends on:*

1. Accuracy $\varepsilon$ (the stricter the accuracy, i.e. the smaller the $\varepsilon$, the more the number of steps).

2. Initial point $\mathbf{x}^{(0)}$ selection, and the initial distance of $f(\mathbf{x}^{(0)})$ from the optimal value (i.e. $\left|f(\mathbf{x}^{(0)}) - f(\mathbf{x}^*)\right|$).

3. *Condition number* $M/m$ (i.e. Parameters $m, M$). The bigger the condition number,the more steps are needed.

**Remark:** If $\frac{m}{M} \ll 1$ we can make the approximation $\log \frac{1}{1-\frac{m}{M}} = -\log(1 - \frac{m}{M}) \overset{\frac{m}{M} \ll 1}{\approx} \frac{m}{M}$
since $\log(1 + x) \approx x$, for $x \ll 1$.

# 6 Newton descent method

Newton method is a special form of descent methods where the descent direction is $D_k = [\nabla^2 f(x^{(k)})]^{-1}$.
The iteration for the Newton descent method is:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - [\nabla^2 f(\mathbf{x}^{(k)})]^{-1} \nabla f(\mathbf{x}^{(k)}) \qquad (40)$$

*Note* that Newton's method converges much faster towards a local minimum (or maximum, if the iteration above is with a "+") than the gradient descent method. Specifically, Newton method converges quadratically.

**Note:** We already have discussed Newton method for one variable in previous sections.

# 7 Selection of step size $\alpha$

We want to find methods to choose the value of step $\alpha^{(k)}$ at each iteration $k$ of the iterative descent algorithm.

If the step is small, we will have little progress from step to step. On the other hand, if the step is large, we will go fast but we will zig-zagging in the solution space.

One idea is to choose the step size so that the value $f\left(\mathbf{x}^{(k+1)}\right)$, in the $(k+1)$-th stage of descent method, is as small as possible.

Some of the most prevalent ways to choose $\alpha^{(k)}$ are the following.

## 7.1 Steepest descent method

The *steepest* descent method is a special case of descent methods, where the step size is selected *such that the value of the objective function at the next iteration is as small as possible*. Assuming a step $\alpha^{(k)}$ and a descent direction $\mathbf{d}^{(k)}$, the step for steepest descent method is found as:

$$\alpha^{(k)} = \arg\min_{\alpha} f(\mathbf{x}^{(k)} + \alpha\,\mathbf{d}^{(k)}) \tag{41}$$

where $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\,\mathbf{d}^{(k)}$ is the $(k+1)$-th point in the descent direction method.

We can view $f(\mathbf{x}^{(k)} - \alpha^{(k)}\mathbf{d}^{(k)})$ as a function of one variable (the step), and its optimum is found by taking the derivative

equal to $0$. Thus, at each iteration, the step is optimized so that it causes the largest decrease in the value of the objective function at iteration $k+1$, $f(\mathbf{x}^{(k+1)})$.

$$\frac{\mathrm{d}f(\mathbf{x}^{(k)} - \alpha^{(k)}\mathbf{d}^{(k)})}{\mathrm{d}\alpha^{(k)}} = 0\,. \tag{42}$$

This method accelerates the search of global minimum $\mathbf{x}^*$ for a function $f$. *Note* that this method is characterized as greedy because it makes the locally optimal choice at each stage with the hope of finding the global optimum faster.

### 7.1.1 Examples

**Example 1**

Consider the quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} - \mathbf{b}^T\mathbf{x}$, where $Q$ is a $n \times n$ symmetric ($Q = Q^T$) positive-definite matrix, $\mathbf{b} \in \Re^n$, $\mathbf{x} \in \Re^n$. We want to find the form of iteration of gradient descent using the steepest descent method.

Thus, we need to find the step $a^{(k)}$ and the gradient $\nabla f(\mathbf{x}^{(k)}) = \mathbf{g}^{(k)}$ at each step, and then the iteration will be:

- $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha^{(k)}\nabla f(\mathbf{x}^{(k)})$.

We have:

- $\nabla f(\mathbf{x}^{(k)}) = Q\mathbf{x}^{(k)} - \mathbf{b} = \mathbf{g}^{(k)}$

- $\alpha^{(k)} = \arg\min_\alpha f(\mathbf{x}^{(k+1)})$
  $= \arg\min_\alpha f(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)})$
  $= \arg\min_\alpha \frac{1}{2}(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)})^T Q(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)}) - \mathbf{b}^T(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)})$

- Define
  $\Phi(\alpha) = \frac{1}{2}(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)})^T Q(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)}) - \mathbf{b}^T(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)})$

- By differentiating $\Phi(\alpha)$ with respect to step size $\alpha$, we get:
  $\Phi'(\alpha) = \frac{1}{2}[(-\mathbf{g}^{(k)})^T Q(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)})]$
  $+\frac{1}{2}[(\mathbf{x}^{(k)} - \alpha\mathbf{g}^{(k)})^T Q(-\mathbf{g}^{(k)})] + \mathbf{b}^T\mathbf{g}^{(k)}$
  $= (\mathbf{x}^{(k)} - t\mathbf{g}^{(k)})^T Q(-\mathbf{g}^{(k)}) + \mathbf{b}^T\mathbf{g}^{(k)}.$
  In order to find $\alpha$ which minimizes $\Phi(\alpha)$, we take the derivative of $\Phi'(\alpha)$ equal to zero:
  $\Phi'(\alpha) = 0$
  $\Leftrightarrow \alpha\mathbf{g}^{(k)^T} Q\mathbf{g}^{(k)} = (\mathbf{x}^{(k)^T} Q - \mathbf{b}^T)\mathbf{g}^{(k)} = 0$
  Observe that:

$$(\mathbf{x}^{(k)^T} Q - \mathbf{b}^T) = \mathbf{g}^{(k)^T} \tag{43}$$

  Then we get that

$$a^{(k)} = \frac{\mathbf{g}^{(k)^T}\mathbf{g}^{(k)}}{\mathbf{g}^{(k)^T} Q\mathbf{g}^{(k)}} = \frac{||\mathbf{g}^{(k)}||^2}{\mathbf{g}^{(k)^T} Q\mathbf{g}^{(k)}}. \tag{44}$$

Finally, the iteration is:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \frac{||\mathbf{g}^{(k)}||^2}{\mathbf{g}^{(k)^T} Q \mathbf{g}^{(k)}} \mathbf{g}^{(k)}. \qquad (45)$$

**Example 2**

Consider function $f(\mathbf{x}) = f(x_1, x_2) = \frac{1}{2}\left(x_1^2 + \gamma x_2^2\right)$, that has minimum value $0$, achieved at $\mathbf{x}^* = (0, 0)$.

This is a quadratic function where $Q = \begin{pmatrix} 1 & 0 \\ 0 & \gamma \end{pmatrix}$ and $b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. We want to find the form iteration of gradient descent using the steepest descent method.

The iteration for each variable will be:

$$x_1^{(k+1)} = x_1^{(k)} - \alpha^{(k)} \nabla f(x_1^{(k)}) = \left(1 - \alpha^{(k)}\right) x_1^{(k)}$$

$$x_2^{(k+1)} = x_2^{(k)} - \alpha^{(k)} \nabla f(x_2^{(k)}) = \left(1 - \gamma\alpha^{(k)}\right) x_2^{(k)}$$

Thus,we need to find step $\alpha^{(k)}$ at each stage.

We have:

$$\begin{aligned}
\alpha^{(k)} &= \arg\min_{\alpha} f(\mathbf{x}^{(k+1)}) \\
&= \arg\min_{\alpha} f(x_1^{(k+1)}, x_2^{(k+1)}) \\
&= \arg\min_{\alpha} \frac{1}{2}(1-\alpha^{(k)})^2(x_1^{(k)})^2 \\
&\quad + \frac{1}{2}\gamma(1-\gamma\alpha^{(k)})^2(x_2^{(k)})^2
\end{aligned}$$

Define:

$$G(\alpha) = \frac{1}{2}\left[(1-\alpha^{(k)})^2(x_1^{(k)})^2 + \gamma(1-\gamma\alpha^{(k)})^2(x_2^{(k)})^2\right].$$

By differentiating $G(\alpha)$ with respect to step size $a$, setting the derivative equal to zero and we find $\alpha^*$, and replacing $\alpha^*$, the iterations become:

$$x_1^{(k+1)} = \gamma\left(\frac{\gamma-1}{\gamma+1}\right)^k \tag{46}$$

and

$$x_2^{(k+1)} = \left(-\frac{\gamma-1}{\gamma+1}\right)^k \tag{47}$$

Actually, we can observe that as $\gamma$ approaches $1$, then $Q$ becomes closer to the unit function, and the minimum and maximum eigenvalue of $Q$ become equal (and equal to $1$ each. Function $f$ is an ellipsoid, which tends to a circle as $\gamma$ goes to $1$. As $\gamma$ goes to $1$ the convergence to the global optimum becomes faster.

## 7.2 The Armijo rule

The Armijo rule for the step is to start with $m_k = 0$, then $m_k$ is set to the first nonnegative integer $m$ for which

$$f(\mathbf{x}^{(k)}+\alpha\cdot\beta^m U^{(k)})-f(\mathbf{x}^{(k)}) \leq \alpha\cdot\beta^m \nabla^T f(\mathbf{x}^{(k)})U^{(k)} , \tag{48}$$

where $\alpha$, $\beta$ are positive constant numbers, with $0 < \beta < 1$, $U^{(k)} = -\nabla f(\mathbf{x}^{(k)})$.

## 7.3 Diminishing step size

The Diminishing step rule is choosing any step size that satisfies the following conditions:

(a) The produced sequence of steps $\{\alpha^{(k)}\} \rightarrow 0$ as $k \rightarrow \infty$

(b) $\sum_{k=1}^{\infty} \alpha^{(k)} = \infty$

This rule is guaranteed to converge to the optimum $\mathbf{x}^*$.

**Note:**

There is also another descent method with constant step $a$ at all iterations, which is simpler to implement but not as efficient as the methods above.

# 8 Convergence Order

We now provide a systematic methodology to study convergence of descent methods.

Consider that $e(\mathbf{x}^{(k)}) = \left\|\mathbf{x}^{(k)} - \mathbf{x}^*\right\|$ or equal to $\left\|f(\mathbf{x}^{(k)}) - f\right.$ denotes the error at iteration $k$. Given a sequence of points generated by the gradient descent algorithm, $\mathbf{x}^{(k)}$, $k = 1, 2, 3...$,we say that it converges **linearly** to vector $\mathbf{x}^*$, if there exists a number $\beta \in (0, 1)$ such that:

$$\lim_{k\to\infty} \frac{e(\mathbf{x}^{(k+1)})}{e(\mathbf{x}^{(k)})} \leq \beta < 1.$$

If the above holds with $\beta = 0$,then is said to converge **superlinearly**. So, for superlinear convergence (which is faster than linear):

$$\lim_{k\to\infty} \frac{e(\mathbf{x}^{(k+1)})}{e(\mathbf{x}^{(k)})} = 0.$$

In superlinear convergence, the sequence defined by ratios of errors in two successive steps must go to zero.

The ***convergence order*** of $\mathbf{x}^{(k)}$ is $p \in \mathcal{R}$ $(1 \leq p \leq \infty)$, if

$$0 < \lim_{k\to\infty} \frac{e(\mathbf{x}^{(k+1)})}{\left[e(\mathbf{x}^{(k)})\right]^p} < \infty$$

Then we get:

$$e(\mathbf{x}^{(k+1)}) \approx c{\cdot}e\left(\mathbf{x}^{(k)}\right)^p \qquad \text{for some} \quad c \in \mathcal{R},\ 0 < c < \infty$$

The larger the $p$, the faster the convergence. It turns out that:

- The Gradient descent method has $p = 1$ (linear convergence)
- The Newton method has $p = 2$ (quadratic convergence), which is faster than the gradient method.

## 8.1 Further issues with Convergence

Any descent method produces a sequence of points $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}$ which converges to minimum $\mathbf{x}^*$ when it reaches a point such that $\nabla f(\mathbf{x}^*) = 0$. This means that the function $f(\mathbf{x})$ does not changes at this point (the point is called **stationary point** of the function) that is, in every subsequent iteration, the value of $\mathbf{x}^{(k+1)}$ will be stationary and equal to $\mathbf{x}^*$.

*What is the problem that arises?*

Is every limit point of the sequence of points $\{\mathbf{x}^{(k)}\}$ generated by the descent method a stationary point?

In some iteration of the algorithm, if the descent direction is almost orthogonal to the gradient vector, then we will have

$$\frac{\nabla^T f(\mathbf{x}^k) d_k}{\left\| f(\mathbf{x}^{(k)}) \right\| \, \|d_k\|} \longrightarrow 0 \tag{49}$$

and the algorithm will therefore stop, while it is $\nabla f(\mathbf{x}^*) \neq 0$. In these cases, the algorithm gets stuck at a limit point

which is *not* a stationary point. These situations arise if I do not choose correctly the descent direction.

In order to avoid this serious problem, we need to put some constraints on how we choose the descent direction, so that $\nabla^T f(\mathbf{x}^k) d_k$ is sufficiently large.

Proposition 1:

Assume that the eigenvalues of $D_k$ are bounded above and also bounded away from zero, Namely, there always exist two number $c_1, c_2 \neq 0$,such that:

$$c_1 \left\|\mathbf{z}\right\|^2 \leq \mathbf{z}^T D_k \mathbf{z} \leq c_2 \left\|z\right\|^2 \ \forall \mathbf{z} \in \mathcal{R}^n \, . \qquad (50)$$

We also have that $\mathbf{z}^T D_k \mathbf{z} = \lambda \left\|\mathbf{z}\right\|^2$, where $\lambda$ denotes any of the eigenvalues of $D_k$, that is:

$$c_1 \left\|\mathbf{z}\right\|^2 \leq \lambda \left\|\mathbf{z}\right\|^2 \leq c_2 \left\|\mathbf{z}\right\|^2 \, . \qquad (51)$$

Thus, if $\mathbf{d}_k = -D_k \nabla f(\mathbf{x}^{(k)})$ and we choose $D_k$ such that its eigenvalues be upper and lower bounded, then:

$$\left|\nabla^T f(\mathbf{x}^{(k)}) \mathbf{d}_k\right| \geq c_1 \left\|\nabla f(\mathbf{x}^{(k)})\right\|^2 \qquad (52)$$

and

$$\left\|d_k\right\|^2 \leq {c_2}^2 \left\|\nabla f(\mathbf{x}^{(k)})\right\|^2 \qquad (53)$$

Since, we ensured that $\nabla^T f(\mathbf{x}^{(k)}) \mathbf{d}_k \neq 0$ and $\left\|\mathbf{d}_k\right\|$ is upper bounded, the problem is no longer exists.

*Note* that if $D_k = I$ (case of Gradient method) then the conditions in the inequalities above are obeyed and $c_1 = c_2 = 1$.

Definition:

The sequence of directions $\{\mathbf{d}_k\}$ is called *gradient-related* to the sequence of the points $\{\mathbf{x}_k\}$, if for every subsequence $\{\mathbf{x}_k\}_{k\in\mathcal{K}}$ that converges to a non-stationary point $\mathbf{x}$ (such that $\nabla f(\mathbf{x}) \neq 0$), the corresponding subsequence $\{d_k\}_{k\in\mathcal{K}}$ is bounded and it satisfies:
$\lim_{k\to\infty_{k\in\mathcal{K}}} \nabla^T f(\mathbf{x}^{(k)})\mathbf{d}_k < 0$.

So, if the conditions above with the eigenvalues of $D_k$ are satisfied, direction $d_k = -D_k\nabla f(\mathbf{x}^{(k)})$ is gradient-related to $\{\mathbf{x}_k\}$.

We now state some additional propositions:

**1**. Consider a sequence of points $\{\mathbf{x}_k\}$ produced by a gradient algorithm which $d_k$ is *gradient-related* and $a_k$ is selected according to the Armijo's rule. Then every *marginal* point of $\{\mathbf{x}_k\}$ is also a *stationary* point (i.e. $\nabla f(\mathbf{x}^{(k)}) = 0.$ )

**2**.Consider a sequence of points $\{\mathbf{x}_k\}$ produced by a gradient algorithm which $\mathbf{d}_k$ is *gradient-related*. Assume also that there exists an $L > 0$ such that:

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\| \ . \qquad (54)$$

If $\alpha_k$ is selected so as to satisfy the following equation:

$$\epsilon < a_k \leq (2-\epsilon)\frac{\left|\nabla^T f(\mathbf{x}^{(k)})\mathbf{d}_k\right|}{L\left\|d_k\right\|^2} \tag{55}$$

**or** according to the diminishing step rule, **then** every *limit* point of $\{\mathbf{x}_k\}$ is also a *stationary* point.

## 8.2 Example

We will now consider the special case of quadratic functions and will study their convergence.

Consider function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x}$ where $Q > 0$ (positive definite), and consider the Gradient descent algorithm for finding the minimum of $f(\mathbf{x})$ (which is $\mathbf{x}^* = \mathbf{0}$. We need to find the optimal value $a_k^*$, so that fast convergence of the algorithm be achieved.

The iteration of the algorithm is:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha^{(k)}Q\mathbf{x}^{(k)} = (I - \alpha^{(k)}Q)\mathbf{x}^{(k)}\,, \tag{56}$$

which is equal to:

$$\|\mathbf{x}^{(k+1)}\|^2 = (\mathbf{x}^{(k)})^T(I - \alpha^{(k)}Q)^2\mathbf{x}^{(k)} \tag{57}$$

Knowing that $\left(\mathbf{x}^{(k)}\right)^T A\mathbf{x}^{(k)} \leq \lambda_{max}(A)\left\|\mathbf{x}\right\|^2$ for nay matrix $A$ and its maximum eigenvalue $\lambda_{max}(A)$, we get:

$$\left\|\mathbf{x}^{(k+1)}\right\|^2 \leq \lambda_{max}(I - \alpha^{(k)}Q)^2\left\|\mathbf{x}^{(k)}\right\|^2 \tag{58}$$

where $\lambda_{max}(I-\alpha^{(k)}Q)^2$ is the maximum eigenvalue of matrix $(I-\alpha^{(k)}Q)^2$.

Observe that:

$$\lambda_{max}(I-\alpha^{(k)}Q)^2 = max\{\left(1-\alpha^{(k)}m^2\right), \left(1-\alpha^{(k)}M^2\right)\}$$

(by using the property that $\lambda(A^2) = \lambda(A)^2$, where $\lambda(A)$ denotes the eigenvalues of matrix $A$). Let $m$,$M$ denote the minimum and maximum eigenvalue of matrix $Q$ respectively, then we get:

$$\frac{\|\mathbf{x}^{(k+1)}\|}{\|\mathbf{x}^{(k)}\|} \leq max\{|1-\alpha^{(k)}m\|, \|1-\alpha^{(k)}M\|\} \quad (59)$$

Since $Q > 0$ then $m, M > 0$. Thus, we have $\frac{\|\mathbf{x}^{(k+1)}\|}{\|\mathbf{x}^{(k)}\|} < 1$, which shows that $\left\|\mathbf{x}^{(k+1)}\right\|$ is decreasing in every step of the algorithm.

Now we want to determine $\alpha^{(k)}$ so that the convergence decreases with maximum rate. We have the problem:

$$\min_{\alpha^{(k)}} \max\{(1-\alpha^{(k)}m), (1-\alpha^{(k)}M)\} \quad (60)$$

Which gives that ${\alpha_k}^* = \frac{2}{M+m}$ and consequently

$$\frac{\left\|\mathbf{x}^{(k+1)}\right\|}{\left\|\mathbf{x}^{(k)}\right\|} \leq \frac{M-m}{M+m}. \quad (61)$$

# 9 The use of gradient method in communication networks

## 9.1 Localization of jammers in wireless networks

Consider a sensor network or any other type of network, where there is wireless communication among sensors, and there is one jammer-sensor(attacker) which transmits intermittent signals to disrupt communication.

The network must have the ability:

1. *Detect* the jammer
2. *Localize* the jammer based only on the network (without adding new infrastructure), i.e estimate its location with good accuracy.
3. *Respond* to the attack (defend against or capture the jammer)

We will address the localization problem. We need to use an inherent property of wireless networks, against which the jammer cannot hide. We note the following:

- In wireless networks the received power is exponentially decreasing with the distance $d$ from the source (transmitter):

$$P_{received}(d) \approx \frac{c \cdot P}{d^a} \text{ for } a \geq 2 \qquad (62)$$

where $P$ is the transmission power of the source and $c$ is a constant.

- For every node, we define a *packet delivery ratio* $PDR = f(SINR)$ which denotes the probability that a transmitted packet is received at the node correctly. Note that the *packet error rate* $PER = f(SINR)$ is the probabilistic complement of $PDR$ (i.e. $PER = 1 - PDR$).

When a node (let it be $node_A$) is closer to the jammer than another node ($node_B$) then the communication of $node_B$ is disrupted by jammer less than the one of $node_A$ , that is $PDR_{node_B} < PDR_{node_A}$. This is because the farther located node receives less power by the jammer.

To localize the jammer with maximum accuracy, we need to find the node which has the lowest PDR. This will be found by a variant of the gradient method.

In this problem the Gradient method corresponds to a kind of routing algorithm which finds the closest node to the jammer. The points are the node positions on the two-dimensional plane.

Suppose an arbitrary node $n_0$ starts the localization process. the node checks his own PDR and asks its neighbors to tell his their PDR. If there exists at least one neighbor with a smallest PDR, then the token is passed to the neighbor with the smallest PDR (kind of steepest descent), say node $n_1$. Then, node $n_1$ does a similar comparison of its PDR with the PDRs

of its neighbors, and so on.

The algorithm stops when every neighbor node has higher $PDF$ than the current node.

**Remark:** Due to the randomness of the wireless channel signal quality (due to fading etc) which is superimposed to the path loss, it may happen that a node that is farther from the source has a larger received signal than a node that is closer to the source. Thus, the algorithm might also have errors in some cases.

## 9.2 Content distribution and caching

Consider a network, that is represented by a graph $G = (V, E)$, with $M$ unit size items. Each node $n$ can receive and store some items and its storage capacity is $K_n$. In a smaller scale, its cache memory can be $K_n$.

An information item request triggers traffic from the requester to the node that has the file. The replication scheme of items across the network is important.

Performance metric: total traffic generated. If node $y$ needs to access a file stored at node $x$, the traffic load is equal to the length of the path from $x$ to $y$.

The operation of the network is as follows:

- *Replication:* Replicates items in different nodes of the network.

For each item $m$ the replication frequency $f_m$ that the item is copied in the network must be determined. *Note* that, for every node $i$ $\sum_{m=1}^{M} x_{mi} \leq K_n$ (i.e. the total size of items that reside at a node cannot exceed its storage capacity).

- There is no reason not to operate the network at full storage capacity, so we assume that there are $K_n$ items stored at each node.

- Access: If a node $\mathbf{x}$ requests an item $m$, it will take $m$ from the closest node that has it.

- For each item $m$ we define set $\mathcal{H}_m$ which denotes the set of all possible configurations of the item in the network. For example, in a network with $n$ nodes, $\mathcal{H}_m$ will be the set of all $n$-dimensional vectors with zeros and ones, where a $1$ in component $i$ means that the item is stored in node $i$, and a $0$ it is not stored in $i$. Of course, the total number of ones in node $i$ must not exceed $K_n$.

Also denote $\mathcal{H}$ the set of all possible caching configurations for all items in the network.

- *Traffic load* for configuration $H \in \mathcal{H}$ is $T(H) = \sum_{m=1}^{n} T(H_m$ where $T(H_m)$ is the traffic load for each item.

- Define ${N_i}^m$: being the set of all nodes that receive item $m$ from node $i$ (i.e. all nodes for which node $i$ is the closest node that has item $m$).

- We consider ${\lambda_j}^m$ the request rate of item $m$ from node $j$.

Thus, we have:

$$T(H_m) = \sum_{i=1}^{f_m} \sum_{j \in {N_i}^m} \lambda_j^m D_{ij} \tag{63}$$

where $D_{ij}$ denotes the distance of node $i$ that has the item $m$ from the node $j$ that asks for it.

In this problem an algorithm that resembles the Gradient method becomes as follows:
Starting from a feasible initial solution $H_0$, create a sequence of configurations $H_0 \rightarrow H_1 \rightarrow H_2 \rightarrow \ldots$ , where in each step we want to minimize the traffic load rearranging the way items are stored (*Heuristic* algorithm). Specifically we swap the pair of items $m, m'$ that reduces the total load with maximal amount.

For each item $m$ in the cache of node $i$:
(a) Compute the increase in traffic if $m$ is removed from the cache (because all requests that access the particular item at node $i$ will need to access it at another node $j$).

(b) For each item $m'$ not in the cache of $i$, compute the decrease in overall traffic load if $m'$ is cached at node $i$ (since some requests will access $m'$ as their closest replica).

If (maximum traffic decrease over all items $m'$) $>$ (minimum traffic increase over all items $m$), then swap $m, m'$. Continue until no further improvements in traffic load can be made. We reach a local minimum in the optimization problem above.

## 9.3 Throughput maximization with power control in wireless communications

**Example:** There are $N$ transmitters, $N$ receivers, and each transmitter $i$ is connected to a receiver $i$.

The Signal to Interference and Noise Ratio at each receiver $i$ as a function of the transmitter power vector $\mathbf{P} = (P_1, \ldots, P_N)$ is given by

$$SINR_i(\mathbf{P}) = \frac{G_{ii}P_i}{\sum_{j=1, j\neq i}^{N} G_{ji}P_j + n_i} \tag{64}$$

where $n_i$ is the thermal noise power at receiver $i$ and $P_i$ is the transmission power of transmitter $i$.

The *capacity* for each link $i$ is
$C_i = \log_2(1 + SINR_i(\mathbf{P})) \approx \log_2(SINR_i(\mathbf{P}))$ for large

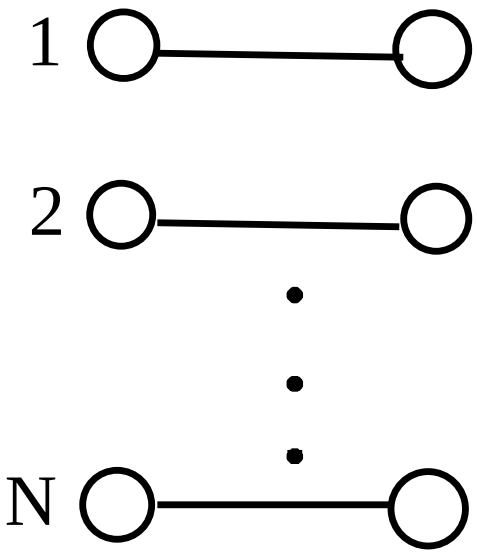


Figure 5: N transmitters connected to N receivers. The different transmitter-receiver links can have different relative positions.

enough SINRs. This denotes the maximum rate at which information can b e transmitted on link $i$ (from Shannon capacity formula in information theory).

We assume that at each transmitter $i$, where packets arrive and wait in a queue before being transmitted. Let $q_i$ be the number of packets (size of queue) that are waiting for transmission. The product $q_i C_i(\mathbf{P})$ is called *throughput* of link $i$.

Note that *throughput* is different than *capacity*. Capacity does not take into account the queue length, it just says how much can be transmitted on the link. If we activate some queue for which the transmitter-receiver link has large capacity, but the

queue is empty, finally no traffic with go through (zero throughput) since the queue is empty.

The queue length changes dynamically with time since the packet arrival processes at each queue is dynamic and the rate at which information is transmitted (packets leave the queue) is also dynamic and depends on the link quality.

**Our purpose:** We wish to maximize the sum of $q_i C_i(\mathbf{P})$, namely the total system throughput, or

$$\max_{\mathbf{P}} \sum_{i=1}^{N} q_i C_i(\mathbf{P}),$$

by appropriately controlling the transmit power vector $\mathbf{P}$.

Thus, we have to optimize the function:

$$\max_{(P_1,\ldots,P_N)} \sum_{l=1}^{N} q_l \log(SINR_l) =$$

$$\max_{(P_1,\ldots,P_N)} \sum_{l=1}^{N} q_l \log\left(\frac{G_{ll}P_l}{\sum_{k=1,k\neq l}^{N} G_{kl}P_k + N_l}\right) =$$

$$\max_{(P_1,\ldots,P_N)} f\big(\mathbf{P}\big)$$

by controlling transmission power vector $\mathbf{P} = (P_1, \ldots, P_N)$, where $q_i$ is the queue size of transmitter $i$ and $C_i(P)$ is the capacity of link $i$.

Although $f(P)$ does not seem to be a concave function of $P$, we will transform it to a concave function. If we prove that $f\left(\mathbf{P}\right)$ is concave the local maximum is global as well.

Consider transformation: $\tilde{P}_l = \ln \mathbf{P}_l$,for $l = 1, \ldots, N$ and the vector $\tilde{P} = (\tilde{P}_1, \ldots, \tilde{P}_N)$.

$$f(\tilde{P}) = \sum_{l=1}^{N} q_l \log \left( \frac{G_{ll} e^{\tilde{P}_l}}{\sum_{k \neq l}^{N} G_{kl} e^{\tilde{P}_k} + N_l} \right) =$$

$$\sum_{l=1}^{N} q_l \Big[ \underbrace{\log \left( G_{ll} e^{\tilde{P}_l} \right)}_{\text{term } 1} - \underbrace{\log \Big( \sum_{k \neq l}^{N} G_{kl} e^{\tilde{P}_k} + N_l \Big)}_{\text{term } 2} \Big]$$

Term 1 is linear in $\tilde{\mathbf{P}}$. We will prove that term 2 is convex in $\tilde{\mathbf{P}}$. In order to do that, we shall examine first under which conditions a function $f(\cdot)$ that arises as composition of functions $h(\cdot)$ and $g(\cdot)$ is convex or concave.

Let $f(x) = (h \circ g)(x)$, with $f(x) = h(g(x))$. For functions of one variable $x$, $h : \mathbf{R} \longrightarrow \mathbf{R}$ and g: $\mathbf{R^n} \longrightarrow \mathbf{R}$, assume that $h$ and $g$ are twice differentiable. In this case convexity of $f(\cdot)$ means $f''(x) \geq 0$ for all $x \in \mathbf{R}$. The first and the second derivatives of $f(\cdot)$, $f = h \circ g$ are:

$$f'(x) = h'(g(x))g'(x),$$

$$f''(x) = h''(g(x))(g'(x))^2 + h'(g(x))g''(x)$$

Function $f$ is concave if: (i.) $h$ is concave, increasing and $g$ is concave, or

(ii.) $h$ is concave, decreasing and $g$ is convex.

Function $f$ is convex if:

(i.) $h$ is convex, increasing and $g$ is convex, or

(ii.) $h$ is convex, decreasing and $g$ is convex.

**Vector Composition:** Consider the case where $f(.)$ is the composition of several functions, that is:

$$f(x) = h(g_1(x), \ldots, g_k(x)) = h(\mathbf{g}(x))$$

with $h : \mathcal{R}^k \longrightarrow \mathcal{R}$, $g_i : \mathbf{R} \longrightarrow \mathbf{R}$.

The first and the second derivatives of $f(.)$ are as follows:

$$f'(x) = \nabla h^T(\mathbf{g}(x))\mathbf{g}'(x)$$

$$f''(x) = \mathbf{g}'^T(x)\nabla^2 h(\mathbf{g}(x))\mathbf{g}'(x) + \nabla h^T(\mathbf{g}(x)\mathbf{g}''(x)$$

where $\mathbf{g}'(x) = (g_1'(x), \ldots, g_k'(x))$. Function $f$ is convex if:

(i.) $h$ is convex, increasing in each argument and $g_i$ is convex.

(ii.) $h$ is convex, decreasing in each argument and $g_i$ is concave.

**Proof of concavity of $f(\tilde{P})$:**

Consider function $h(\mathbf{z}) = \log(\sum_{i=1}^{k} e^{z_i})$. As a first step for

proving concavity of $f(\tilde{P})$, we will prove that $h(\mathbf{z})$ is convex, or that its Hessian matrix $H(\mathbf{z}) > 0$.

The first derivative of the function $h(\mathbf{z})$ is:

$$\frac{\vartheta h(\mathbf{z})}{\vartheta z_i} = \frac{e^{z_i}}{\sum_{k=1}^{k} e^{z_i}}$$

The second derivative with respect to the $i$-th component:

$$\frac{\vartheta^2 h(\mathbf{z})}{\vartheta z_i^2} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} - \frac{e^{2z_i}}{(\sum_{j=1}^{k} e^{z_j})^2}$$

The second derivative with respect to component $z_i, z_j$ with $(i \neq j)$ is:

$$\frac{\vartheta^2 h(\mathbf{z})}{\vartheta z_i \vartheta z_j} = -\frac{e^{z_i + z_j}}{(\sum_{j=1}^{k} e^{z_i})^2}$$

Consider the case of $N = 2$ to better visualize the situation. Let $A = e^{z_1} + e^{z_2}$. Then for any $\mathbf{v} \geq 0$, $\mathbf{v} = (v_1, v_2)$ the quadratic for $\mathbf{v}^H H \mathbf{v}$, with $H$ the Hessian of $h(z)$, should be shown non-negative. Thus:

$$\mathbf{v}^T H \mathbf{v} \geq 0 \Leftrightarrow$$

$$\Leftrightarrow \begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} \frac{Ae^{z_1}-e^{2z_1}}{A^2} & -\frac{e^{z_1+z_2}}{A^2} \\ -\frac{e^{z_1+z_2}}{A^2} & \frac{Ae^{z_2}-e^{2z_2}}{A^2} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \geq 0$$

$$\Leftrightarrow v_1^2 \left( \frac{e^{z_1}}{A} - \frac{e^{2z_1}}{A^2} \right) - 2v_1 v_2 \frac{e^{z_1+z_2}}{A^2} + v_2^2 \left( \frac{e^{z_2}}{A} - \frac{e^{z_2}}{A^2} \right) =$$

$$= \frac{A(v_1^2 e^{z_1} + v_2^2 e^{z_2}) - (v_1 e^{z_1} + v_2 e^{z_2})^2}{A^2} \geq^{?} 0. \quad (65)$$

In order to prove the above, we use the Cauchy-Schwartz inequality for vectors $\mathbf{q}, \mathbf{b}$:

$$(\mathbf{a}^T \mathbf{a})(\mathbf{b}^T \mathbf{b}) \geq (\mathbf{a}^T \mathbf{b})^2$$

for vectors

$$\mathbf{a} = \begin{pmatrix} e^{\frac{z_1}{2}} & e^{\frac{z_2}{2}} \end{pmatrix}, \mathbf{b} = \begin{pmatrix} v_1 e^{\frac{z_1}{2}} & v_2 e^{\frac{z_2}{2}} \end{pmatrix}.$$

Once we proved that inequality (3) holds, we have proved that $h(\mathbf{z}) = \log(\sum_{i=1}^{k} e^{z_i})$ is convex, $h(.)$ is increasing ($\nearrow$) in its argument $z_i$. Thus, function $h(\mathbf{g}(x)) = \log(\sum_{i=1}^{k} e^{g_i(x)})$ is convex if $g_i(x)$ is convex [rule (i.)].

Now, we have $f(\tilde{P}$ equals to:

$$\sum_{l=1}^{N} q_l \Bigg[ \underbrace{\log\left( G_{ll} e^{\tilde{P}_l} \right)}_{\text{term 1}} - \underbrace{\log\left( \sum_{k \neq l} e^{\ln G_{kl} + \tilde{P}_k} + N_l \right)}_{\text{term 2}} \Bigg]$$

Term 1 is linear in $\tilde{\mathbf{P}}$, as we said before. Also, $g_k(x) = \ln G_{kl} + \tilde{P}_k$ is linear in $\tilde{\mathbf{P}}_k$, so it is convex as well. Thus, $\log \sum_{k \neq l} e^{\ln G_{kl} + \tilde{P}_k} + N_l$ is convex in $\tilde{\mathbf{P}}$ and thus $\big( - \log \sum_{k \neq l} e^{\ln G_{kl} + \tilde{P}_k} + N_l \big)$ is concave and the whole $f(\tilde{\mathbf{P}})$ is concave in $\tilde{\mathbf{P}}$.

Now, we use the gradient *ascent* method to find the global maximum of $f(\mathbf{P})$ (we come back to the initial notation with $\mathbf{P}$, since we have used the transformation to $\tilde{\mathbf{P}}$ only to show the concavity of $f(\cdot)$).

$$f(\mathbf{P}) = \sum_{l=1}^{N} q_l \log \frac{G_{ll} P_l}{\sum\limits_{k \neq l}^{N} G_{kl} P_k + N_l}$$

$$\frac{\vartheta f(\mathbf{P})}{\vartheta P_l} = \frac{q_l}{P_l} - \sum_{j \neq l} \frac{q_j G_{jl}}{\sum_{k \neq j} G_{jk} P_k + N_j}$$

1. Start with initial vector $\mathbf{P}^{(0)} = (P_1^{(0)}, \ldots, P_N^{(0)})$.

2. At the $(k+1)$-th step of the algorithm, we have iteration: $\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} + \beta \nabla f(\mathbf{P}^{(k)})$, where $\beta$ is the (say) constant step size.

Note that since we want to maximize $f(\mathbf{P})$, we have a gradient ascent method, with $f(\mathbf{P}^{(k+1)}) > f(\mathbf{P}^{(k)})$ and we move

towards the direction of maximum increase of $f(\cdot)$, (i.e towards the direction of the gradient).

Each transmitter $l$ updates its power according to rule:

$$P_l^{(t+1)} = P_l^{(t)} + \beta \frac{\vartheta f P}{\vartheta P_l} \Longrightarrow$$

$$\Longrightarrow P_l^{(t+1)} = P_l^{(t)} + \beta \left( \frac{q_l}{P_l^{(t)}} - \sum_{j \neq l} \underbrace{\frac{q_j G_{il}}{\sum_{k \neq j} G_{jk} P_k^{(t)} + N_j}}_{m_j^{(t)}} \right)$$

Let $m_j^{(t)}$ given as noted above. By multiplying numerator and denominator by $G_{jj}$ and $P_j^{(t)}$ we have:

$$m_j^{(t)} = \frac{q_j SINR_j^{(t)}}{G_{jj} P_j^{(t)}}$$

Consequently, at the $(t+1)$-th step of the iteration we have at the $\ell$-th transmitter:

$$P_l^{(t+1)} = P_l^{(t)} + \beta \left( \frac{q_l}{P_l^{(t)}} - \sum_{j \neq l} G_{jl}\, m_j^{(t)} \right)$$

Thus $m_j^{(t)}$ can be considered as a message pertaining to transmitter $j$, for $j = 1, \ldots, N$. Each transmitter knows its queue

$q_j$, its gain to its receiver $G_{jj}$ and its transmission power $P_j^{(t)}$. It also receives channel state information (CSI) from the receiver in the form of SINR$_j$.

Each node $j$ broadcasts this message to every other node, $l \neq j$ which then updates its power according to the rule above.

It turns out that the gradient ascent method $\mathbf{P}^{(t+1)} = \mathbf{P}^{(t)} + \beta \nabla f(\mathbf{P}^{(t)})$ above (for any initial power vector $\mathbf{P}^{(0)}$ and any sequence in which the iteration is executed by users) converges to the optimal vector $\mathbf{P}^*$. This is the vector that maximizes $f(\mathbf{P})$.

The algorithm is fully distributed, since each transmitter autonomously uses quantities that he knows and needs to know only messages $m_j(t)$. Each transmitter knows $q_i$, $P_i$, while it can get the $SINR$ at its corresponding receiver (the receiver measures it and sends it back to transmitter). Each transmitter also obtains quantities (messages) $m_j(t)$ from other nodes $j \neq i$.

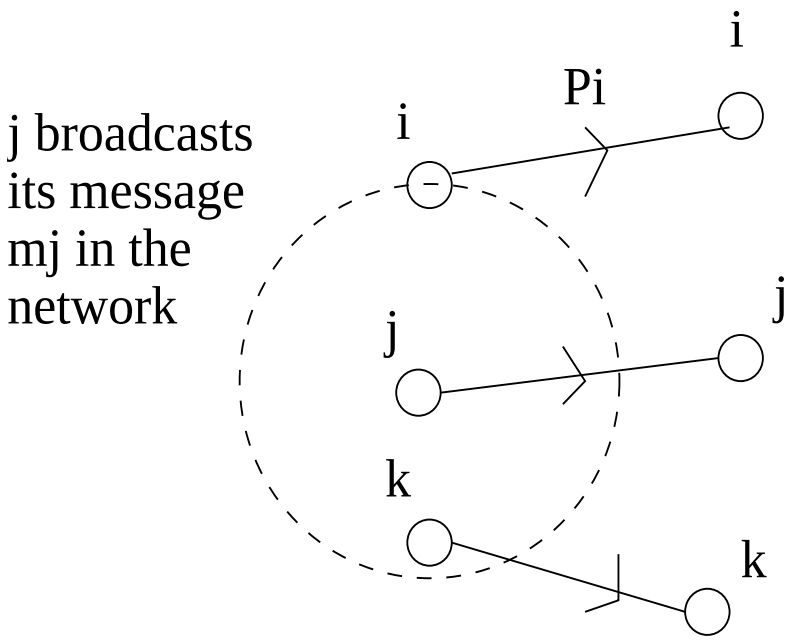


Figure 6: Explanation of message broadcasting in the network for distributed algorithm operation.

# 10 Other remarks on gradient descent method

## 10.1 Distributed implementation

In general, if the objective function is separable in its variables, namely if

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n)$$

can be written as sum of functions, where each function depends only on one variable, i.e if

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} \underbrace{f_i(x_i)}_{concave}$$

then the iteration of gradient ascent method

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \beta \nabla f(\mathbf{x}^{(t)})$$

becomes:

$$x_i^{(t+1)} = x_i^{(t)} + \beta \frac{\vartheta f_i(x_i)}{\vartheta x_i}$$

and thus there is no message passing needed among nodes. Simply, each node computes $\frac{\vartheta f_i(x_i)}{\vartheta x_i}$ (since it knows $f_i(x_i)$, but not $f_j(x_j)$) and does the updates of its variables independently from others.

The independent iterations for each node will again lead to the optimal vector $\mathbf{x}^*$, namely the vector that maximizes $f(\mathbf{x})$.

## 10.2 Gauss-Seidel method and Jacobi methods

When, in every step, the update of each variable is done *serially* then we have the **Gauss-Seidel** method:

$$x_i^{(t+1)} = \arg\min_{x_i} f\left(x_1^{(t+1)}, \ldots, x_{i-1}^{(t+1)}, x_i, x_{i+1}^{(t)}, \ldots, x_n^{(t)}\right)$$

When in each step the update of variables is computed *simultaneously*, and then the updates are combined in order to find the optimal solution, then we have the **Jacobi** method:

$$x_i^{(t+1)} = \arg\min_{x_i} f\left(x_1^{(t)}, \ldots, x_{i-1}^{(t)}, x_i, x_{i+1}^{(t)}, \ldots, x_n^{(t)}\right)$$

At each time iteration $t$, a vector corresponding to the new point is found, in the direction of a certain partial derivative, and then all these vectors are combined to find the point $\mathbf{x}^{(t)}$.

# References


[1] Boyd and Vandenberghe, ”Convex Optimization”

[2] Bertsekas, ”Non Linear Programming”