Lecture 9

9.1 Strongly Convex Optimization

We have proved the following rate of convergence, for the fast gradient method as applied to a convex function f with Lipschitz continuous gradient, $x_k = \mathrm{FGM}_k(f, x_0)$ for $k \geq 0$ iterations, we have

$$f(x_k) - f^\star \quad \leq \quad \tfrac{2L\|x_0 - x^\star\|^2}{k^2}, \qquad k \geq 1.$$

Now, assume that the objective is additionaly strongly convex with parameter $\mu > 0$. Then, we have the following bound:

$$\tfrac{\mu}{2}\|x_0 - x^\star\|^2 \quad \leq \quad f(x_0) - f^\star.$$

Combining these two inequalities together, we get:

$$f(x_k) - f^\star \quad \leq \quad \tfrac{4L}{\mu k^2}(f(x_0) - f^\star).$$

Let us set

$$K \quad := \quad \sqrt{\tfrac{8L}{\mu}} \tag{9.1}$$

and run the fast gradient method for this number of iterations. As a result we halve the functional residual:

$$f(x_K) - f^\star \quad \leq \tfrac{1}{2}(f(x_0) - f^\star).$$

Now, we can restart this procedure again (starting a new fresh version of the fast gradient method from the output of the previous run). We perform the following iterations, starting from $y_0 := x_0$:

$$\boxed{y_{t+1} \quad = \quad \mathrm{FGM}_K(f, y_t), \qquad t \geq 0,} \tag{9.2}$$

and we need $T = \log_2 \frac{f(x_0) - f^\star}{\varepsilon}$ restarts in order to obtain obtain $f(y_T) - f^\star \leq \varepsilon$.

Theorem 9.1.1. *The total complexity of the fast gradient method with restarts is*

$$\sqrt{\tfrac{8L}{\mu}} \log_2 \tfrac{f(x_0) - f^\star}{\varepsilon} \tag{9.3}$$

first-order oracle calls to minimize a strongly convex smooth function.

This is the same optimal dependence on the condition number $\sqrt{\frac{L}{\mu}}$ as we obtained for the heavy ball method. However, the fast gradient method works for a larger class of all smooth functions (not necessary quadratic).

Using a more advanced reasoning it is possible to obtain the same complexity on the strongly convex functions without restarts.

Exercise 9.1.1. Develop a direct version of the fast gradient method for strongly convex functions (without restarts), that achieves the optimal dependence (9.3) on the problem class parameters.

Note that to perform restarts, we need to know the condition number (9.1), which is not a trivial knowledge in practice.

- In a direct version of the fast gradient method that takes into account strong convexity, we still need to know the constant of strong convexity $\mu \geq 0$.
- We may avoid knowing L by performing an adaptive search, analogously to that one for the basic gradient method.

9.2 Applications: Machine Learning

9.2.1 Generalized Linear Models

Consider the following objective, for a loss function $\ell : \mathbb{R} \to \mathbb{R}$:

$$f(x) \quad = \quad \tfrac{1}{m}\sum_{i=1}^{m} \ell(\langle a_i, x\rangle - b_i) \tag{9.4}$$

and the following optimization problem to train our model:

$$\min_{x\in\mathbb{R}^n} \Big[f(x) + \psi(x) \Big],$$

where $\psi(\cdot)$ is usually some regularizer. $a_1, \ldots, a_m \in \mathbb{R}^n$ and $b_1, \ldots, b_m \in \mathbb{R}$ are given data, and $x \in \mathbb{R}^n$ represents parameters of the model that we want to learn.

Loss Functions. We assume loss $\ell(\cdot)$ to be convex and differentiable. Classical examples of loss functions are:

- *Quadratic loss:* $\ell(t) = \frac{1}{2}t^2$, then the objective f is a quadratic multivariate function. It has the Lipschitz derivative with constant $L_\ell = 1$.
- *Logistic loss:* $\ell(t) = \log(1 + e^t)$. It has the Lipschitz derivative with constant $L_\ell = \frac{1}{4}$.
- *Huber loss:*

$$\ell(t) \quad = \quad \begin{cases} \frac{1}{2\delta}t^2, & -\delta \leq \tau \leq \delta, \\ |t| - \frac{\delta}{2}, & \text{otherwise,} \end{cases}$$

  which is a smooth approximation of the absolute valuer $|t|$, and it has the Lipschitz derivative with constant $L_\ell = \frac{1}{\delta}$.

Choice of the Norm. It is convenient to analyze objective (9.4) in a matrix form. Denote by $g : \mathbb{R}^m \to \mathbb{R}$ the separable loss:

$$g(y) \quad = \quad \tfrac{1}{m}\sum_{i=1}^{m} \ell(y^{(i)}).$$

Then, g has a Lipschitz gradient with respect to the standard Euclidean norm, with constant $L_g = L_\ell$. We can form our data into the matrix $A \in \mathbb{R}^{m\times n}$ and the vector $b \in \mathbb{R}^m$. Then, our objective (9.4) is

$$\begin{aligned} f(x) \quad &= \quad g(Ax + b), \\ \nabla f(x) \quad &= \quad A^\top \nabla g(Ax + b). \end{aligned} \tag{9.5}$$

We have two basic choices for the norm:

- Fix the standard Euclidean norm in $\mathbb{R}^n$, $\|x\| = \langle x, x \rangle^{1/2}$. Then, $L = L_\ell \cdot \|A\|^2$ with respect to this norm. Then, the main step of the fast gradient method reads as:

$$v_{k+1} \;=\; v_k - a_{k+1} \nabla f(y_k),$$

  and it requires to perform two matrix-vector products per iteration (one with matrix A and another with matrix $A^\top$).

- Fix the generalized Euclidean norm with $B = A^\top A \in \mathbb{R}^{n \times n}$, that is $\|x\| = \langle Bx, x \rangle^{1/2}$. When the number of data is large ($m \gg n$), we have $B \succ 0$. In practice, we can always us a regularized matrix, $B = A^\top A + \delta I$ for a small $\delta > 0$.

  The Lipschitz constant is much smaller with respect to this norm: $L = L_\ell$ (for $\delta = 0$), and in all cases above it is just an absolute constant that does not depend on the data! It is easy to check that the fast gradient method can be deduced entirely identical for a generalized Euclidean norm. Then, the main iteration of the fast gradient method is the *preconditioned* gradient step:

$$v_{k+1} \;=\; v_k - a_{k+1} B^{-1} \nabla f(y_k).$$

  Therefore, we have to invert the matrix B, but we need to do it only once in the beginning before running an algorithm. However, this preconditioning significantly improves overall performance and it also provides us with a clear way of choosing the Lipschitz constant.

Efficient Implementation. Note that in the fast gradient method, we choose

$$y_k \;=\; \gamma_k v_k + (1 - \gamma_k) x_k, \tag{9.6}$$

where $\gamma_k \in (0, 1)$ is our parameter that depends on a_{k+1} and $A_{k+1} = A_k + a_{k+1}$ as follows: $\gamma_k = \frac{a_{k+1}}{A_{k+1}}$. At the same time, according to our theory, an optimal choice for a_{k+1} that solves the quadratic equation is:

$$a_{k+1} \;=\; \frac{1}{2L} \cdot \Bigl(1 + \sqrt{1 + 4A_k L}\Bigr).$$

Now imagine that we choose L in this formula adaptively, possible performing several different tries per iteration. Than, for each try of $\bar{L}$ we have to compute $\bar{a}_{k+1}$, $\bar{\gamma}_k$, and the corresponding gradient $\nabla f(\bar{y}_k)$ at the intermediate point (9.6). Each new computation of the gradient would involve two matrix-vector products, if implemented straightforwardly. However, notice that due to the structure (9.5),

$$\begin{aligned} \nabla f(\bar{y}_k) \;&=\; \nabla f(\bar{\gamma}_k v_k + (1 - \bar{\gamma}_k) x_k) \\ &\overset{(9.5)}{=}\; A^\top \nabla g(\bar{\gamma}_k A v_k + (1 - \bar{\gamma}_k) A x_k + b). \end{aligned} \tag{9.7}$$

We see that we do not need to recompute $A\bar{y}_k$ each time: it is enough compute Av_k and Ax_k once, and then use them to construct $A\bar{y}_k$ for all different $\bar{\gamma}_k$. This will save us few matrix-vector products (overall, it can make the method 2-5 times faster!)

The same technique can be applied for a line search used in the classic gradient descent. Imagine we want to perform the classic gradient update: $x^+ = x - \frac{1}{\bar{L}} \nabla f(x)$ for different $\bar{L} > 0$, checking the following inequality:

$$f(x) - f(x^+) \;\geq\; \frac{1}{2\bar{L}} \|\nabla f(x)\|^2.$$

If implemented straightforwardly, we need to recompute $f(x^+)$ possibly several times per iteration. However, we notice that

$$f(x^+) \overset{(9.5)}{=} g(Ax^+ + b) = g(Ax - \tfrac{1}{\bar{L}}A\nabla f(x) + b).$$

Hence, having computed Ax and $A\nabla f(x)$ once, we can evaluate the new function value $f(x^+)$ very efficiently for many different values of $\bar{L}$. This implementation trick makes it really efficient when solving large-scale problems.

Regularizers. When training a model, we typically solve the following optimization problem,

$$\min_{x\in\mathbb{R}^n} \Big[f(x) + \psi(x) \Big], \tag{9.8}$$

where f is the main part of our objective, and ψ is some *simple regularizer*. The most common examples include:

- *ℓ_2-regularization.* $\psi(x) = \frac{\mu}{2}\|x\|^2$, where $\mu > 0$ is the regularization parameter.

  This makes our objective *strongly convex* and therefore we will always have a unique global solution, and our methods will exhibit fast *linear convergence rates.* We do not need to change anything in the gradient method, as it will automatically adjust to strong convexity. On the contrary, we need to modify the fast gradient method, taking $\mu > 0$ into account (either performing restarts, or a modified choice of parameters).

- *ℓ_1-regularization.* $\psi(x) = \lambda\|x\|_1$, where $\lambda > 0$. This is popular to induce desired sparsity in the solution $x^\star$. Note that full objective (9.8) is still convex, but it becomes non-differentiable. Therefore we cannot technically apply our smooth methods anymore (as we simply cannot compute gradients). Later in the course we will study general methods that can be applied for non-smooth convex optimization. However, typically these methods are much slower than those ones for smooth convex optimization (the complexity becomes $O(\frac{1}{\varepsilon^2})$ instead of $O(\frac{1}{\varepsilon^{1/2}})$, where ε is the target accuracy in terms of the functional residual).

  Luckily, there is a very efficient approach to properly *modify the step* of the basic and the fast gradient methods, which will ensure the same fast convergence rates. We will study this modification later in this lecture in the most general form.

- *Simple constraints.* A related to the previous case, a typical situation is when there are additional *simple* convex constraints $Q \subset \mathbb{R}^n$ that we want to induce into our problem (e.g. that the parameter variable lie in a given ball, a box, a simplex, etc.). This can be modeled by the following artificial function (which is still convex):

  $$\psi(x) = \begin{cases} 0, & x \in Q, \\ +\infty, & \text{otherwise.} \end{cases}$$

  Then, problem (9.8) becomes constrained optimization problem:

  $$\min_{x\in Q} f(x).$$

  The modification that will follow from our general construction will be simply to add *projection* after the main step of the fast gradient method:

  $$v_{k+1} = \pi_Q(v_k - a_{k+1}\nabla f(y_k)). \tag{9.9}$$

  Everything else (including the fast rates of convergence!) remain the same. "Simplicity" of the set Q means that we can actually perform projection (9.9) efficiently.

9.2.2 Nonlinear Models: Neural Networks

In non-linear models (e.g. deep neural networks), we replace the affine part $Ax - b$ in our objective by a nonlinear operator. Therefore, the objective gets the following form:

$$f(x) \;\; = \;\; \tfrac{1}{m}\sum_{i=1}^{m} \ell(m_i(x)), \tag{9.10}$$

where $m_i(\cdot)$ represent the output of the model on i-th training example, for a given value of parameters. In case of the simplest neural networks, it has the form of composition of $T \geq 1$ linear layers:

$$m_i(x) \;\; = \;\; \langle x^{(T)}, \; \sigma(\ldots X^{(3)}\sigma(X^{(2)}\sigma(X^{(1)}a_i)))\rangle,$$

where $X^{(1)}, \ldots, X^{(T-1)}$ are matrices of parameters of appropriate shapes, and $x^{(T)}$ is a vector (if we want the result of the last layer to be a number), which are all stacked together into

$$x \;\; = \;\; \Bigl[X^{(1)}, X^{(2)}, \ldots, X^{(T-1)}, x^{(T)}\Bigr],$$

and $\sigma(\cdot)$ is a point-wise nonlinear function (typically, either *relu activation* $\sigma(t) = \max\{0, t\}$, *sigmoid*, $\sigma(t) = \frac{1}{1+e^{-t}}$, or *hyperbolic tangent*, $\sigma(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$.

In case where $T = 1$ (one layer), we recover the linear models. Note that, in general ($T \geq 2$), the objective in (9.10) is non-convex and therefore no longer belongs to our problem class. However, both Nesterov's accelerated method and Polyak's heavy-ball method are widely applied in practice and remain the de facto standards for incorporating momentum into training algorithms.

Investigating the properties of objective functions such as (9.10) (e.g., *hidden convexity*) and the dynamics of first-order algorithms applied to them (*overparametrization*, *implicit bias*, *the edge of stability*) remains an active area of research in theoretical deep learning.

9.3 Fully Composite Problems

9.3.1 Motivation

Now, we have a perfect picture about the problem class of smooth convex functions (at least in high dimension $n \to \infty$). We discussed the lower bounds for this problem class and the fast gradient method that is *optimal* (thus its upper bound on the complexity matches the lower bound up to a numerical constant).

This class is reach enough, as it includes, for example, objectives of the form (9.4). Now, before switching to some entirely different situations, we might ask the following question:

how much we can extend our current problem class such that the fast gradient method still works?

We will still work with convex objectives, as convexity was one of the crucial building blocks of the analysis of the fast gradient method. However, what we want is to include at least problems with simple constraints and simple regularizers. As a result, we might allow our objective to be *non-smooth*, but the non-smoothness should be controlled explicitly by our structure.

We want to be abstract enough, as to be able to cover as biggest mathematical formulation of the optimization problem as possible, for which the fast gradient method is still applied. Besides practical importants (as to cover many interesting applications), such generalization is very instructive: as the formulation becomes more abstract, we can use only the very basic mathematical operations (convexity, monotonicity), and it allows us to distinguish the most important steps of the analysis.

9.3.2 Fully Composite Formulation

Now we consider the following optimization problem

$$\min_{x \in Q} \varphi(x), \tag{9.11}$$

where φ has the following *fully composite structure*:

$$\varphi(x) \quad = \quad F(x, f_1(x), \ldots, f_m(x)), \qquad x \in Q \subseteq \mathbb{R}^n.$$

Set Q is convex (that is, with any two points $x, y \in Q$ it contains entire segment between them: $\lambda x + (1-\lambda)y \in Q$ for any $0 \leq \lambda \leq 1$).

Smooth Components. Now, instead of one main objective function $f : \mathbb{R}^n \to \mathbb{R}$ we have m "smooth components" $f_1, \ldots, f_m$. Each of these functions $f_i : Q \to \mathbb{R}$ is convex and has a Lipschitz continuous gradient with constant $L^{(i)} > 0$, for every $1 \leq i \leq m$:

$$\|\nabla f_i(x) - \nabla f_i(y)\| \quad \leq \quad L^{(i)} \|x - y\|, \qquad x, y \in Q.$$

We combine all Lipschitz constants into one vector $L = [L^{(1)}, \ldots, L^{(m)}] \in \mathbb{R}^m$. We can all stack all these functions into one vector function $f : Q \to \mathbb{R}^m$:

$$f(x) \quad = \quad \Bigl[f_1(x), \ldots, f_m(x)\Bigr]^\top \quad \in \quad \mathbb{R}^m.$$

These functions are the "difficult parts" of the problem. Thus, we only assume a black-box access to the first-order oracles. With abuse of notation, we denote by $\nabla f(x) \in \mathbb{R}^{m \times n}$ the Jacobian of the mapping f at point $x \in Q \subseteq \mathbb{R}^n$, that is simply composed by the gradient of all f_i:

$$\nabla f(x) \quad = \quad \begin{bmatrix} \nabla f_1(x)^\top \\ \nabla f_2(x)^\top \\ \cdots \\ \nabla f_m(x)^\top \end{bmatrix} \quad \in \quad \mathbb{R}^{m \times n}.$$

Composite Component. With the vector notation, we can write down our target objective as

$$\min_x \Bigl[\varphi(x) \quad = \quad F(x, f(x)),\Bigr] \tag{9.12}$$

where $F : Q \times \mathbb{R}^m \to \mathbb{R}$ is the "outer" or "composite" component of the objective. The main assumption about F is that it is *simple enough.* Namely, we assume that we can solve the following optimization subproblems efficiently:

$$\min_x F(x, Ax + b) + \tfrac{\alpha}{2}\|x\|^2, \tag{9.13}$$

with an arbitrary affine mapping. That is the linearized version of the original problem.

Moreover, we will need the following formal assumptions about F:

1. F is a *jointly convex* function on $Q \times \mathbb{R}^m$
2. F is *monotone* in the second argument: for any $u, v \in \mathbb{R}^m$ s.t. $u \leq v$ (coordinate-wise), it holds: $F(x, u) \leq F(x, v)$ for any $x \in Q$. Monotonicity ensures that φ as the composition of F and f will be convex.
3. We also assume that F is Lipschitz in second argument:

   $$|F(x, u) - F(x, v)| \quad \leq \quad M\|u - v\|, \qquad x \in Q, u, v \in \mathbb{R}^m,$$

   for some constant $M > 0$.

Examples.

1. *Classical unconstrained minimization*: we have only one function $f_1(x)$ and $F(x,u) \equiv u^{(1)}$. Then,
$$\varphi(x) \;=\; F(x, f(x)) \;\equiv\; f_1(x).$$
The subproblem (9.13) that we require to be able to solve efficiently is the simplest quadratic minimization, as for computing the gradient step, for a certain $a \in \mathbb{R}^n$ and $\alpha > 0$:
$$\min_x \Big\{ \langle a, x \rangle + \tfrac{\alpha}{2}\|x\|^2 \Big\}$$

2. *Non-smooth regularization*: set $F(x,u) \equiv u^{(1)} + \psi(x)$, where ψ is a given convex (possibly non-smooth) function. Then,
$$\varphi(x) \;=\; F(x, f(x)) \;\equiv\; f(x) + \psi(x).$$
This example covers both constrained minimization and ℓ_1-regularization. The subproblem (9.13) becomes:
$$\min_x \Big\{ \langle a, x \rangle + \tfrac{\alpha}{2}\|x\|^2 + \psi(x) \Big\}$$
and is often called as the proximal operator for ψ.

3. *Max-type problems*: let $F(x,u) \equiv \max\limits_{1 \le i \le m} u^{(i)}$. Then,
$$\varphi(x) \;=\; F(x, f(x)) \;\equiv\; \max_{1 \le i \le m} f_i(x). \tag{9.14}$$
Note that such functions will always be convex (for convex components f_i), but non-smooth. Such objective might appear, for example, if we want to solve the *feasibility problem* with a convex set given by functional inequalities:
$$x^\star \;\in\; Q \;=\; \Big\{ x \in \mathbb{R}^n \;:\; f_1(x) \le 0, \ldots, f_m(x) \le 0 \Big\}. \tag{9.15}$$
Then, we can aim to solve (9.15) by minimizing objective of the form (9.14). As we saw in the first lecture, feasibility problems are very general and are equivalent to minimization problems.

   In our method as applied to solve (9.14), we require to compute a solution to the following non-smooth convex subproblem with the explicit structure:
$$\min_x \Big\{ \max_{1 \le i \le m} \Big[\langle a_i, x \rangle - b_i \Big] + \tfrac{\alpha}{2}\|x\|^2 \Big\}.$$
   There are efficient solvers that can be applied to solve this subproblem in general.

9.3.3 Composite Fast Gradient Method

Now, we present the fully composite version of the fast gradient method, as to solve problems of the form (9.12). As before, we generate two sequences of points: the main sequence $\{x_k\}_{k \ge 0}$ and the sequence of auxiliary points $\{v_k\}_{k \ge 0}$, both starting from the same initialization, $x_0 = v_0 \in \operatorname{dom} \varphi$. We use a sequence of growing coefficients $\{A_k\}_{k \ge 0}$ starting from $A_0 = 0$.

Algorithm 9.1: *Fully Composite Fast Gradient Method.*

```
Initialization: $x_0 \in \mathrm{dom}\,\varphi$. Set $v_0 = x_0$ and $A_0 = 0$. Fix $K \geq 1$.
For $k = 0 \dots K-1$ iterate:
    1. Choose a new coefficient $a_{k+1} > 0$. Set $A_{k+1} := A_k + a_{k+1}$ and $\gamma_k := \frac{a_{k+1}}{A_{k+1}}$
    2. Compute the function values $f(y_k) \in \mathbb{R}^m$ and the Jacobian $\nabla f(y_k) \in \mathbb{R}^{m \times n}$ at the
       intermediate point $y_k := \gamma_k v_k + (1-\gamma_k) x_k$
    3. Compute the new auxiliary point $v_{k+1}$ by solving the following linearized subproblem:
       $v_{k+1} = \arg\min_x \Big[ F(x, f(y_k) + \nabla f(y_k)(x - y_k)) + \frac{1}{2a_{k+1}} \|x - v_k\|^2 \Big]$
    4. Set a new point from the triangle rule: $x_{k+1} := \gamma_k v_{k+1} + (1-\gamma_k) x_k$
Return $x_K$
```

9.3.4 Analysis

Surprisingly, the analysis of Algorithm 9.1 almost identically repeats the analysis of the basic version of the fast gradient method. We only need to be careful when working with the composite outer part $F(\cdot, \cdot)$.

Our goal is to prove by induction the following inequality, for any $k \geq 0$:

$$\tfrac{1}{2}\|x - x_0\|^2 + A_k \varphi(x) \quad \geq \quad \tfrac{1}{2}\|x - v_k\|^2 + A_k \varphi(x_k), \qquad x \in \mathrm{dom}\,\varphi. \tag{9.16}$$

It obviously holds for $k = 0$. Assume that it holds for some $k \geq 0$ and consider one step of the method. We have:

$$\begin{aligned}
\tfrac{1}{2}\|x - x_0\|^2 + A_{k+1}\varphi(x) \quad &= \quad \tfrac{1}{2}\|x - x_0\|^2 + a_{k+1}\varphi(x) + A_k \varphi(x_k) \\
&\overset{(9.16)}{\geq} \quad \tfrac{1}{2}\|x - v_k\|^2 + a_{k+1}\varphi(x) + A_k\varphi(x_k) \\
&= \quad \tfrac{1}{2}\|x - v_k\|^2 + a_{k+1}F(x, f(x)) + A_k\varphi(x_k) \\
&\geq \quad \tfrac{1}{2}\|x - v_k\|^2 + a_{k+1}F(x, f(y_k) + \nabla f(y_k)(x - y_k)) + A_k\varphi(x_k),
\end{aligned} \tag{9.17}$$

where in the last inequality we used convexity of each f_i, in a vector component-wise form:

$$f(x) \quad \geq \quad f(y_k) + \nabla f(y_k)(x - y_k) \quad \in \quad \mathbb{R}^m,$$

and the monotonicity of F in the second argument.

Note that by definition, $v_{k+1} = \arg\min_x m_k(x)$ is the minimum of the strongly convex model from the right hand side of (9.17):

$$m_k(x) \quad := \quad \tfrac{1}{2}\|x - v_k\|^2 + a_{k+1}F(x, f(y_k) + \nabla f(y_k)(x - y_k)) + A_k\varphi(x_k).$$

Hence, we have

$$m_k(x) \quad \geq \quad \tfrac{1}{2}\|x - v_{k+1}\|^2 + m_k^\star,$$

where

$$\begin{aligned}
m_k^\star &= m_k(v_{k+1}) \\
&= \tfrac{1}{2}\|v_{k+1} - v_k\|^2 + a_{k+1}F(v_{k+1}, f(y_k) + \nabla f(y_k)(v_{k+1} - y_k)) + A_k F(x_k, f(x_k)) \\
&\overset{(*)}{\geq} \tfrac{1}{2}\|v_{k+1} - v_k\|^2 + a_{k+1}F(v_{k+1}, f(y_k) + \nabla f(y_k)(v_{k+1} - y_k)) \\
&\qquad + A_k F(x_k, f(y_k) + \nabla f(y_k)(x_k - y_k)) \\
&\overset{(**)}{\geq} \tfrac{1}{2}\|v_{k+1} - v_k\|^2 + A_{k+1}F(x_{k+1}, f(y_k) + \nabla f(y_k)(x_{k+1} - y_k)) \\
&= \tfrac{1}{2\gamma_k^2}\|x_{k+1} - y_k\|^2 + A_{k+1}F(x_{k+1}, f(y_k) + \nabla f(y_k)(x_{k+1} - y_k)),
\end{aligned}$$

where we used in $(*)$ again convexity of f and the monotonicity of F, and in $(**)$ we used the joint convexity of F.

Now, let us use that components of f have the Lipschitz continuous gradients, in a vector component-wise form:

$$0 \leq f(x_{k+1}) - f(y_k) - \nabla f(y_k)(x_{k+1} - y_k) \leq \tfrac{\|x_{k+1}-y_k\|^2}{2}L \in \mathbb{R}^m, \tag{9.18}$$

and the fact that $F(\cdot,\cdot)$ is Lipschitz in its second argument:

$$F(x_{k+1}, u_1) \leq F(x_{k+1}, u_2) + M\|u_1 - u_2\|, \qquad u_1, u_2 \in \mathbb{R}^m. \tag{9.19}$$

Hence, we get

$$\begin{aligned}
\varphi(x_{k+1}) &= F(x_{k+1}, f(x_{k+1})) \\
&\overset{(9.19)}{\leq} F(x_{k+1}, f(y_k) + \nabla f(y_k)(x_{k+1} - y_k)) + M\|\delta\|,
\end{aligned}$$

where

$$\|\delta\| := \|f(x_{k+1}) - f(y_k) - \nabla f(y_k)(x_{k+1} - y_k)\| \overset{(9.18)}{\leq} \tfrac{\|x_{k+1}-y_k\|^2}{2}\|L\|.$$

Therefore, combining these observations together, it is sufficient to choose $\frac{1}{\gamma_k^2 A_{k+1}} \geq M\|L\|$ in order to ensure:

$$m_k^\star \geq A_{k+1}\varphi(x_{k+1}).$$

Thus, we established (9.16) for all $k \geq 1$.

By plugging $x := x^\star$ into (9.16) and choosing a_{k+1} by solving the corresponding quadratic equation, we prove the following result.

Theorem 9.3.1. *Let at each iteration of Algorithm 9.1. we choose*

$$a_{k+1} := \tfrac{1}{2\alpha}\Big(1 + \sqrt{1 + 4A_k\alpha}\Big), \quad \textit{where} \quad \alpha := M\|L\|.$$

Then,

$$\varphi(x_k) - \varphi^\star \leq \tfrac{2\alpha\|x_0 - x^\star\|^2}{k^2}, \qquad k \geq 1.$$

Therefore, the fully composite version of the fast gradient method exhibits the same rate of $O(1/k^2)$ as the version for unconstrained minimization from the previous lecture. However, the cost of each step becomes more expensive as it requires solving a non-trivial subproblem with the composite outer part $F(\cdot,\cdot)$.