# Overview of Algorithms

Oscar Dalmau
dalmau@cimat.mx

Centro de Investigación en Matemáticas
CIMAT A.C. Mexico

Feb 2018

# Outline

1. Algorithm overview
2. Convergence order

# Example

**Example 1.1**

Minimize $f(x, y) = xe^{-x^2-y^2}$

# Example

## Example

Minimize $f(x,y) = xe^{-x^2-y^2}$

Gradient:

$$\nabla f(x,y) = e^{-x^2-y^2}\begin{bmatrix} 1-2x^2 \\ -2xy \end{bmatrix}$$

Stationary points:

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} \pm\frac{\sqrt{2}}{2} \\ 0 \end{bmatrix}$$

# Example

## Example

Minimize $f(x,y) = xe^{-x^2-y^2}$

Hessian:

$$\nabla^2 f(x,y) \quad = \quad e^{-x^2-y^2} \left[ \begin{array}{cc} 2x(2x^2-3) & 2y(2x^2-1) \\ 2y(2x^2-1) & 2x(2y^2-1) \end{array} \right]$$

# Example

## Example

Minimize $f(x,y) = xe^{-x^2-y^2}$

Hessian at $[x^*, y^*]^T = [\frac{\sqrt{2}}{2}, 0]^T$

$$\nabla^2 f(x,y) = \sqrt{\frac{2}{e}} \begin{bmatrix} -2 & 0 \\ 0 & -1 \end{bmatrix}$$

then is al local maximum!

# Example

## Example

Minimize $f(x,y) = xe^{-x^2-y^2}$

Hessian at $[x^*, y^*]^T = [-\frac{\sqrt{2}}{2}, 0]^T$

$$\nabla^2 f(x,y) \quad = \quad \sqrt{\frac{2}{e}} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

then is al local minimum!

# Example

**Example 1.2**

Minimize $f(x, y) = x^2 + y^2 + e^{x+y}$

# Example

## Example

Minimize $f(x, y) = x^2 + y^2 + e^{x+y}$

Gradient:

$$\nabla f(x, y) = \begin{bmatrix} 2x + e^{x+y} \\ 2y + e^{x+y} \end{bmatrix}$$

Stationary points: solve the following system of equation!!

$$\begin{bmatrix} 2x + e^{x+y} \\ 2y + e^{x+y} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

It requieres a numerical method!

# General Framework

- Algorithms for unconstrained minimization are iterative methods that find an approximate solution.
- All algorithms for unconstrained minimization require the user to supply a starting point, which we usually denote by $\boldsymbol{x}_0$.
- The user with knowledge about the application and the data set may be in a good position to choose $\boldsymbol{x}_0$ to be a reasonable estimate of the solution.
- Otherwise, the starting point must be chosen by the algorithm, either by a systematic approach or in some arbitrary manner.

# General Framework

- Starting at $\boldsymbol{x}_0$, optimization algorithms generate a sequence of iterates $\{\boldsymbol{x}_k\}_{k=0}^{\infty}$ that terminate when either no more progress can be made or when it seems that a solution point has been approximated with sufficient accuracy.
- In deciding how to move from one iterate $\boldsymbol{x}_k$ to the next, the algorithms use information about the function $f$ at $\boldsymbol{x}_k$ , and possibly also information from earlier iterates $\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_{k-1}$.
- They use this information to find a new iterate $\boldsymbol{x}_{k+1}$ with a lower function value than $\boldsymbol{x}_k$.

# General Framework

1. Start at $\boldsymbol{x}_0$, $k = 0$
2. While not converge
   - Find $\boldsymbol{x}_{k+1}$ such that $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$
   - $k = k + 1$
3. Return $\boldsymbol{x}^* = \boldsymbol{x}_k$

# General Framework: Comment

- However, there exist non-monotone algorithms in which $f$ does not decrease at every step, but $f$ should decrease after some number $m$ of iterations that is, $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_{k-j})$ for some $j \in \mathcal{M} = \{0, 1, \cdots, M\}$ with $M = m - 1$ if $k \geq m - 1$ otherwise $M = k$.
  For example, select $x_{k+1} = x_k + \alpha d_k$, $d_k = -g(x_k)/\|g(x_k)\|$ if

$$f(x_k + \alpha d_k) < \max_{j \in \mathcal{M}} f(\boldsymbol{x}_{k-j}) + \gamma \alpha g(x_k)^T d_k$$

**Note**: See details, in Grippo86NonMonotoneLineSearch.pdf, in internet download Grippo86.pdf

# General Framework

1. How to choose $\boldsymbol{x}_0$?
2. Find a convergence or stop criteria?
3. How to update $\boldsymbol{x}_{k+1}$?

# Updating formula

The algorithm chooses a direction $\boldsymbol{d}_k$ and searches along this direction from the current iterate $\boldsymbol{x}_k$ for a new iterate with a lower function value (line search strategy).

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha \boldsymbol{d}_k$$

# Descent direction

**Definition 1.3**

A descent direction is a vector $\boldsymbol{d} \in \mathbb{R}^n$ such that $f(\boldsymbol{x} + t\boldsymbol{d}) < f(\boldsymbol{x})$, $t \in (0, T)$ i.e., allows to move a point $\boldsymbol{x}$ closer towards a local minimum $\boldsymbol{x}^*$ of the objective function $f : \mathbb{R}^n \to \mathbb{R}$.

There are several methods that compute descent directions, for example: use gradient descent, conjugate gradient method.

# Descent direction

## Descent direction

If $g(\boldsymbol{x})^T \boldsymbol{d} < 0$ then $\boldsymbol{d}$ is a descent direction.

There exists $\hat{t}$ such that $g(\boldsymbol{x} + t\boldsymbol{d})^T \boldsymbol{d} < 0$ for all $t \in [0, \hat{t}]$ (sign preserving theorem).
Using Taylor, there exist $\tau \in (0, 1)$ such that

$$f(\boldsymbol{x} + \hat{t}\boldsymbol{d}) \;\; = \;\; f(\boldsymbol{x}) + \hat{t}\boldsymbol{g}(\boldsymbol{x} + \tau\hat{t}\boldsymbol{d})^T \boldsymbol{d}$$

as $0 < t = \tau\hat{t} < \hat{t}$ then $\boldsymbol{g}(\boldsymbol{x} + \tau\hat{t}\boldsymbol{d})^T \boldsymbol{d} = \boldsymbol{g}(\boldsymbol{x} + t\boldsymbol{d})^T \boldsymbol{d} < 0$ and therefore $f(\boldsymbol{x} + \hat{t}\boldsymbol{d}) < f(\boldsymbol{x})$ then $\boldsymbol{d}$ is a descent direction.

# Line search methods

**Line search methods**

- First, the algorithm chooses a direction $\boldsymbol{d}_k$
- Then, it searches along this direction from the current iterate $\boldsymbol{x}_k$ for a new iterate with a lower function value. The distance to move along $\boldsymbol{d}_k$ can be found by approximately solving the following one-dimensional minimization problem to find a step length $\alpha$:

$$\alpha_k = \arg\min_{\alpha>0} f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)$$

# Search directions for line search method

**The steepest descent direction**

For example $\boldsymbol{d}_k = -\boldsymbol{g}(\boldsymbol{x}_k)$ is the most obvious choice for search direction

- The *steepest descent method* is a line search method that moves along $\boldsymbol{d}_k = -\boldsymbol{g}(\boldsymbol{x}_k)$ at every step.
- Line search methods may use search directions other than the steepest descent direction.
- In general, any descent direction, one that makes an angle of strictly less than $\pi/2$ radians with $-\boldsymbol{g}(\boldsymbol{x}_k)$, is guaranteed to produce a decrease in $f$, i.e. if $\boldsymbol{g}(\boldsymbol{x}_k)^T\boldsymbol{d}_k < 0$ then $|\angle(\boldsymbol{g}(\boldsymbol{x}_k), \boldsymbol{d}_k)| > \pi/2$ , i.e. $|\angle(-\boldsymbol{g}(\boldsymbol{x}_k), \boldsymbol{d}_k)| < \pi/2$, due to $\boldsymbol{g}(\boldsymbol{x}_k)^T\boldsymbol{d}_k = \|\boldsymbol{g}(\boldsymbol{x}_k)\|\|\boldsymbol{d}_k\| \cos\angle(\boldsymbol{g}(\boldsymbol{x}_k), \boldsymbol{d}_k)$.

# Newton direction

- Another important search direction, perhaps the most important one of all, is the *Newton direction*.
- This direction is derived from the second-order Taylor series approximation to $f(\boldsymbol{x}_k + \boldsymbol{d})$

$$f(\boldsymbol{x}_k + \boldsymbol{d}) \quad \approx \quad f(\boldsymbol{x}_k) + \boldsymbol{g}(\boldsymbol{x}_k)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \mathbf{H}(\boldsymbol{x}_k)\boldsymbol{d} \stackrel{def}{=} m_k(\boldsymbol{d})$$

**Newton direction**

$\nabla_{\boldsymbol{d}} m_k(\boldsymbol{d}) = 0$ then $\boldsymbol{d}_k^N = -\mathbf{H}(\boldsymbol{x}_k)^{-1}\boldsymbol{g}(\boldsymbol{x}_k)$ if there exists $\mathbf{H}(\boldsymbol{x}_k)^{-1}$

# Newton direction

- The Newton direction can be used in a line search method when $\mathbf{H}(\boldsymbol{x}_k)$ is positive definite.
- Most line search implementations of Newton's method use the unit step $\alpha = 1$
- When $\mathbf{H}(\boldsymbol{x}_k)$ is not positive definite, the Newton direction may not even be defined, since $\mathbf{H}(\boldsymbol{x}_k)^{-1}$ may not exist.
- Even when it is defined, it may not satisfy the descent property $\boldsymbol{g}_k^T \boldsymbol{d}_k^N < 0$, in which case it is unsuitable as a search direction. In these situations, line search methods modify the definition of $\boldsymbol{d}_k$ to make it satisfy the descent condition.

# Quasi-Newton methods

- Quasi-Newton methods are alternatives to Newton's methods which do not require computation of the Hessian.
- Instead of the true Hessian $\mathbf{H}_k$ , they use an approximation $\mathbf{B}_k$, which is updated after each step.

$$m_k(\boldsymbol{d}) \quad \stackrel{def}{=} \quad f(\boldsymbol{x}_k) + \boldsymbol{g}(\boldsymbol{x}_k)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \mathbf{B}_k \boldsymbol{d}$$

# Quasi-Newton methods

- The approximation $\mathbf{B}_k$ to the Hessian is updated by using successive gradient vectors $\boldsymbol{g}_k, \boldsymbol{g}_{k+1}$ and positions $\boldsymbol{x}_k, \boldsymbol{x}_{k+1}$.
- Quasi-Newton methods are a generalization of the secant method to find the root of the first derivative for multidimensional problems.

# Quasi-Newton methods

- **Recall**: The secant method is defined by the recurrence relation

$$\begin{aligned} x_{k+1} &= x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \\ x_{k+1} &= x_k - \frac{f(x_k)}{f'(x_k)}, \text{ (Newton)} \\ f'(x_k) &\approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}, \text{ (Finite difference)} \end{aligned}$$

- Therefore, the *secant method* can be interpreted as a method in which the derivative is replaced by an approximation and then is a *Quasi-Newton method*.

# Quasi-Newton methods

- Using Taylor Theorem, for the gradient function

$$\nabla f(\boldsymbol{x}+\boldsymbol{h}) \;=\; \nabla f(\boldsymbol{x}) + \nabla^2 f(\boldsymbol{x})\boldsymbol{h} + o(\|h\|)$$

defining $\boldsymbol{x}_k = \boldsymbol{x}$ and $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{h}$ then $\boldsymbol{h} = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and

$$\nabla f(\boldsymbol{x}_{k+1}) \;\approx\; \nabla f(\boldsymbol{x}_k) + \nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$$

- Te previous approximation yields the known *secant equation* that should satisty $\mathbf{B}_k$ (approximation of $\mathbf{H}_k$)

$$\begin{aligned}
\mathbf{B}_{k+1}\boldsymbol{s}_k &= \boldsymbol{y}_k \\
\boldsymbol{s}_k &= \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \\
\boldsymbol{y}_k &= \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)
\end{aligned}$$

# General descent direction

**Descent direction**

If $\mathbf{A}_k$ is any positive definite matrix and $\boldsymbol{g}(\boldsymbol{x}_k) \neq \mathbf{0}$ then $\boldsymbol{d}_k = -\mathbf{A}_k \boldsymbol{g}(\boldsymbol{x}_k)$ is a descent direction

# Convergence order

- Algorithms may differ significantly in their *computational efficiency*.
- A *fast or efficient algorithm* is one that requires only a small number of iterations to converge to a solution and the amount of computation is small.
- In general, in application one uses (or tries to use) the most efficient algorithm.
- How to measure *the rate of convergence* or the efficiency of the algorithms? .
- The most basic criterion is the *order of convergence* of a sequence.