1 Background – unconstrained optimization

An optimization problem is a problem where a cost function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ needs to be minimized or maximized. We will mostly refer to minimization, as maximization can always be achieved by minimizing $-f$. The function $f()$ is called the "objective". In most cases, we wish to find a point $\mathbf{x}^*$ that minimizes f. We will denote this by

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}). \tag{1}$$

This optimization problem is not the most general problem we will deal with. This problem is called an **unconstrained optimization problem**, and we will deal with this problem first. We will deal with constrained optimization in the next section.

Optimization problems arise in huge variety of applications in fields like natural sciences, engineering, economics, statistics and more. Optimization arises from our nature - we usually wish to make things better in some sense. In recent years, large amounts of data have been collected, and consequently optimization methods arise from the need to characterize this data by computational models. In this course we will deal with continuous optimization problems, as opposed to discrete optimization which is a completely different field with different challenges.

Until now we met some optimization problems, such as least squares and $A-$norm minimization problems. Those problems had quadratic objectives, and setting their gradient to zero resulted in a linear system. Understanding quadratic optimization problems are the key for understanding general optimization problems, and often, the iterative methods used for general optimization problems are just a generalization of methods for solving linear systems. We've seen a bit of this concept in Steepest Descent and Gauss-Seidel. Another strong connection between quadratic and general optimization problems is that every objective function f is *locally quadratic*. We will now see that via the Taylor expansion.

1.1 Multivariate Taylor expansion

Assume a continuous one-variable function $f(x)$. The one-dimensional Taylor expansion is given by

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \frac{1}{2}f''(x)\varepsilon^2 + \frac{1}{3!}f'''(c)\varepsilon^3; \quad c \in [x, x+\varepsilon].$$

Assuming that f is continuous, we will usually refer to the last term just as $O(\varepsilon^3)$, and all of our derivations will focus on the first two or three terms. If f has two variables, $f = f(x_1, x_2)$, then the two-dimensional Taylor expansion is given by

$$f(x_1+\varepsilon_1, x_2+\varepsilon_2) = f(x_1, x_2) + \frac{\partial f}{\partial x_1}\varepsilon_1 + \frac{\partial f}{\partial x_2}\varepsilon_2 + \frac{1}{2}\frac{\partial^2 f}{\partial x_1^2}\varepsilon_1^2 + \frac{\partial^2 f}{\partial x_1 x_2}\varepsilon_1\varepsilon_2 + \frac{1}{2}\frac{\partial^2 f}{\partial x_2^2}\varepsilon_2^2.$$

Let us recall the definition of the gradient, which in two-variables is given by

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}.$$

This is a 2×1 vector, and if we denote by $\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2]^\top$, then

$$\langle \nabla f, \boldsymbol{\varepsilon} \rangle = \frac{\partial f}{\partial x_1}\varepsilon_1 + \frac{\partial f}{\partial x_2}\varepsilon_2.$$

Now we will define the two-dimensional Hessian matrix – a two-dimensional second derivative:

$$\nabla^2 f = H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}.$$

It can be seen that $\frac{1}{2}\langle \boldsymbol{\varepsilon}, H\boldsymbol{\varepsilon} \rangle = \frac{1}{2}\frac{\partial^2 f}{\partial x_1^2}\varepsilon_1^2 + \frac{\partial^2 f}{\partial x_1 \partial x_2}\varepsilon_1\varepsilon_2 + \frac{1}{2}\frac{\partial^2 f}{\partial x_2^2}\varepsilon_2^2$, and therefore the Taylor expansion can be written as

$$f(\mathbf{x}+\boldsymbol{\varepsilon}) = f(\mathbf{x}) + \langle \nabla f, \boldsymbol{\varepsilon} \rangle + \frac{1}{2}\langle \boldsymbol{\varepsilon}, H\boldsymbol{\varepsilon} \rangle + O(\|\boldsymbol{\varepsilon}\|^3),$$

This expansion is also suitable for $n-$dimensional functions, where the Hessian matrix $H \in \mathbb{R}^{n\times n}$ is defined by

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

Note that if we assume that f is smooth and twice differentiable, then H is a symmetric matrix (but not necessarily positive definite).

נתונה פונקציה f הבאה,

$$f:\mathbb{R}^2 \to \mathbb{R}$$

$$x \triangleq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$f(x) = 10x_1 + x_1x_2 + e^{x_1-2x_2}$$

חשבו את הגרדיאנט של f.

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 10 + x_2 + e^{x_1-2x_2} \\ x_1 - 2e^{x_1-2x_2} \end{bmatrix}$$

חשבו את ההסיאן של הפונקציה f,

$$H(x) = \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} \end{bmatrix} = \begin{bmatrix} e^{x_1-2x_2} & 1-2e^{x_1-2x_2} \\ 1-2e^{x_1-2x_2} & 4e^{x_1-2x_2} \end{bmatrix}$$

נשים לב כי זו מטריצה סימטרית (בגלל שהנגזרות המעורבות מתחלפות[1]) ולכן היא תמיד לכסינה והע"ע שלה ממשיים.

A Taylor expansion of a function vector In some cases, derivatives may be complicated and we need to know to calculate a derivative of a vector of functions, as opposed to scalar functions that we dealt with so far. Consider $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$. That is $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^m$. The first order approximation of each of the functions $f_i(\mathbf{x})$ is given by

$$f_i(\mathbf{x} + \boldsymbol{\varepsilon}) \approx f_i(\mathbf{x}) + \langle \nabla f_i(\mathbf{x}), \boldsymbol{\varepsilon} \rangle.$$

For the whole function vector $\mathbf{f}(\mathbf{x})$, we can write

$$\delta\mathbf{f} = \mathbf{f}(\mathbf{x}+\boldsymbol{\varepsilon}) - \mathbf{f}(\mathbf{x}) \approx \mathbf{J}\boldsymbol{\varepsilon},$$

where $\mathbf{J} \in \mathbb{R}^{m\times n}$ is the Jacobian matrix, comprised of the gradients $\nabla f_i(\mathbf{x})$ as rows:

$$\mathbf{J} = \begin{bmatrix} - & \nabla f_1(\mathbf{x}) & - \\ - & \nabla f_2(\mathbf{x}) & - \\ & \vdots & \\ - & \nabla f_m(\mathbf{x}) & - \end{bmatrix} \quad \mathbf{J}_{i,j} = \frac{\partial f_i}{\partial x_j}.$$

For the sake of gradient calculation we will focus on the case where $\boldsymbol{\varepsilon} \to 0$ and assume equality in the definition of $\delta\mathbf{f}$.

Example 1. *(The Jacobian of $A\mathbf{x}$) Suppose that $\mathbf{f} = A\mathbf{x}$, where $A \in \mathbb{R}^{m\times n}$. It is clear that*

$$\delta\mathbf{f} = \mathbf{f}(\mathbf{x}+\boldsymbol{\varepsilon}) - \mathbf{f}(\mathbf{x}) = A(\mathbf{x}+\boldsymbol{\varepsilon}) - A\mathbf{x} = A\boldsymbol{\varepsilon}.$$

It is clear that $\mathbf{J} = A$.

Example 2. *(The Jacobian of $\phi(\mathbf{x})$, where ϕ is a scalar function) Suppose that $\mathbf{f} = \phi(\mathbf{x})$, where $\phi : \mathbb{R} \to \mathbb{R}$ is a scalar function, i.e., $f_i(\mathbf{x}) = \phi(x_i)$. In this case the vector Taylor expansion is just a vector of one dimensional expansions.*

$$\delta\mathbf{f} = \phi(\mathbf{x}+\boldsymbol{\varepsilon}) - \phi_i(\mathbf{x}) \approx diag(\phi'(\mathbf{x}))\boldsymbol{\varepsilon} = diag(\phi'(\mathbf{x}))\delta\mathbf{x}.$$

This means that $\mathbf{J} = diag(\phi'(\mathbf{x}))$, which is a diagonal matrix such that $\mathbf{J}_{ii} = \phi'(x_i)$.

1.2 Optimality conditions for unconstrained optimization

We we ask ourselves: what is a minimum point of f? We will consider two kinds of minimum points.

Definition 1 (Local minimum)**.** *A point $\mathbf{x}^*$ will be called a local minimum of a function $f()$ if there exists $r > 0$ s.t*

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) \text{ for all } \mathbf{x} \text{ s.t } \|\mathbf{x}-\mathbf{x}^*\| < r.$$

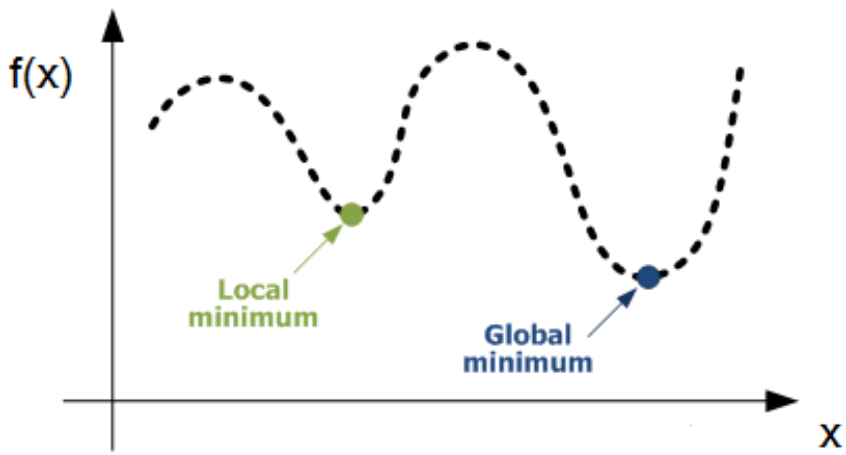

Figure 1: Global and local minimum

If we use a strict inequality, then we have a strict local minimum.

Definition 2 (Global minimum)**.** *A point* $\mathbf{x}^*$ *will be called a global minimum of a function* $f()$ *if*

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) \text{ for all } \mathbf{x} \in \mathbb{R}^n.$$

Again, if we have a strict inequality, we will also have a strict global minimum.

We will ask ourselves now, how can we identify or compute a local and global minima of a function? It turns out that a local minima is possible to define, and a global one is harder. In this course will focus on iterative methods to find a local minimum, hoping that we are not so far from the global minimum. There are many ways to handle issue of global vs. local minimum in the case of general functions—e.g., adding regularization, and investigating a good starting point—but we will not deal with that in this course.

Theorem (Second-Order Sufficient Conditions).

Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.

PROOF. Because the Hessian is continuous and positive definite at x^*, we can choose a radius $r > 0$ so that $\nabla^2 f(x)$ remains positive definite for all x in the open ball $\mathcal{D} = \{z \mid \|z - x^*\| < r\}$. Taking any nonzero vector p with $\|p\| < r$, we have $x^* + p \in \mathcal{D}$ and so

$$\begin{aligned} f(x^* + p) &= f(x^*) + p^T \nabla f(x^*) + \tfrac{1}{2} p^T \nabla^2 f(z) p \\ &= f(x^*) + \tfrac{1}{2} p^T \nabla^2 f(z) p, \end{aligned}$$

where $z = x^* + tp$ for some $t \in (0, 1)$. Since $z \in \mathcal{D}$, we have $p^T \nabla^2 f(z) p > 0$, and therefore $f(x^* + p) > f(x^*)$, giving the result. □

It turns out that there is a quite large family of functions, where any local minimum is also a global minimum. These functions are called convex functions.

1.3 Convexity

The concept of convexity is fundamental in optimization; it implies that the problem is benign in several respects. The term *convex* can be applied both to sets and to functions.

$S \in \mathbb{R}^n$ is a *convex set* if the straight line segment connecting any two points in S lies entirely inside S. Formally, for any two points $x \in S$ and $y \in S$, we have $\alpha x + (1 - \alpha) y \in S$ for all $\alpha \in [0, 1]$.

f is a *convex function* if its domain is a convex set and if for any two points x and y in this domain, the graph of f lies below the straight line connecting $(x, f(x))$ to $(y, f(y))$ in the space $\mathbb{R}^{n+1}$. That is, we have

$$f(\alpha x + (1 - \alpha) y) \leq \alpha f(x) + (1 - \alpha) f(y), \qquad \text{for all } \alpha \in [0, 1].$$

The convex set definition is illustrated in Fig. in The convex function definition can be illustrated in 1D–see Figure 3. A 2D example of convex and non convex function is given in Fig. 4.

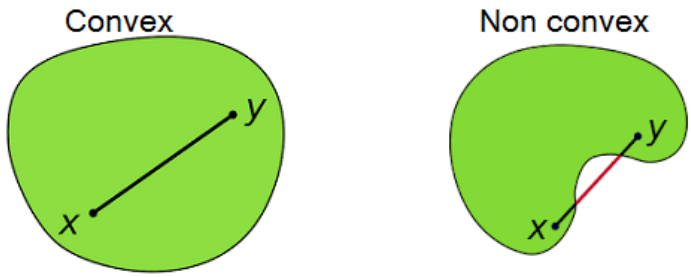

Figure 2: Convexity of sets

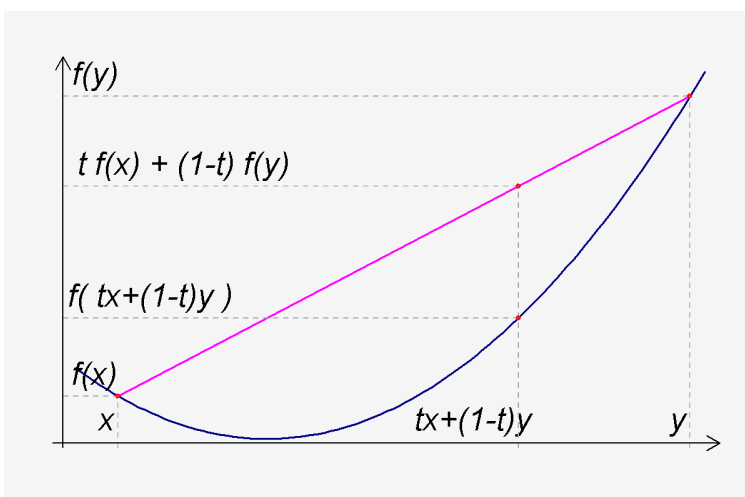

Figure 3: Convexity: the image uses t instead of α

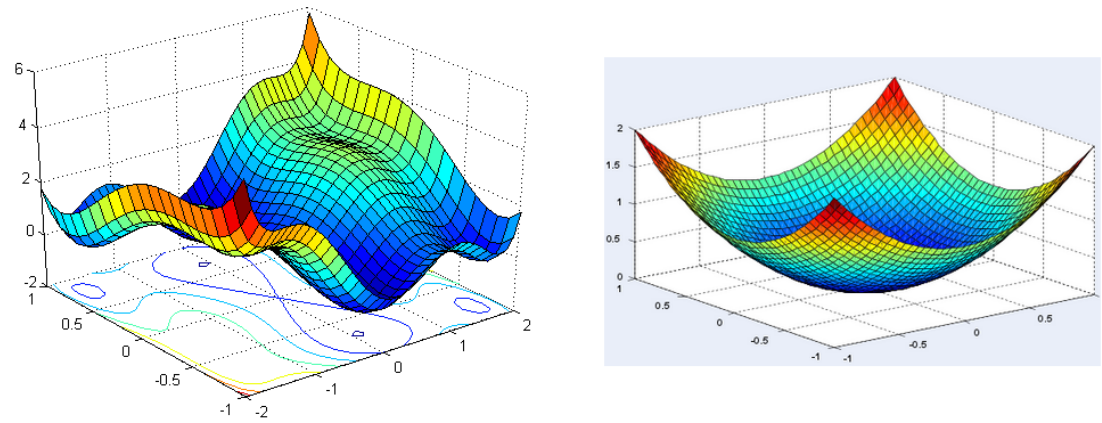

Figure 4: Convexity in 2D: the left image shows a non-convex function, while the right one shows a convex (and quadratic) function.

Alternative definitions for convexity

1. Assume that $f(\mathbf{x})$ is differentiable. $f(\mathbf{x})$ is a convex function over a convex region Ω if and only if for all $\mathbf{x}_1, \mathbf{x}_2 \in \Omega$

$$f(\mathbf{x}_1) \geq f(\mathbf{x}_2) + \langle \nabla f(\mathbf{x}_2), \mathbf{x}_1 - \mathbf{x}_2 \rangle.$$

   This definition means that any tangent plane of the function has to be under the function. This definition is equivalent to the previous traditional definition.

2. Assume that $f(\mathbf{x})$ is twice differentiable. $f(\mathbf{x})$ is a convex function over a convex region Ω if and only if the Hessian $\nabla^2 f(\mathbf{x}) \succeq 0$ is positive semi-definite for all $\mathbf{x}$. It is easy to show that the two definitions are equivalent through a Taylor series:

$$f(\mathbf{x}_1) = f(\mathbf{x}_2) + \langle \nabla f(\mathbf{x}_2), \mathbf{x}_1 - \mathbf{x}_2 \rangle + \frac{1}{2} \langle \mathbf{x}_1 - \mathbf{x}_2, \nabla^2 f(c)(\mathbf{x}_1 - \mathbf{x}_2) \rangle, \quad c \in [\mathbf{x}_1, \mathbf{x}_2].$$

Example 3 (The definitions of convexity for a quadratic function)**.** *A*

נתונה הפונקציה הריבועית

$$f(x)=\frac{1}{2}x^TAx+b^Tx$$

כאשר $A\in S^n$. האם הפונקציה קמורה ותחת איזה תנאים? בדקו לפי שלושת הקריטריונים.

<u>פתרון</u>

בשאלה זו $\Omega=\mathbb{R}^n$.

א).
נבדוק תחת איזה תנאים מתקיים האי"ש:
מתקבל כי, $x_1,x_2\in\Omega$

$$\forall\alpha\in[0,1],x_1,x_2\in\mathbb{R}^n:\ f(\alpha x_1+(1-\alpha)x_2)\le\alpha f(x_1)+(1-\alpha)f(x_2)$$

אגף ימין של המשוואה הוא

$$\alpha f(x_1)+(1-\alpha)f(x_2)=\alpha\left(\frac{1}{2}x_1^TAx_1+b^Tx_1\right)+(1-\alpha)\left(\frac{1}{2}x_2^TAx_2+b^Tx_2\right)=$$
$$=\frac{1}{2}\alpha x_1^TAx_1+\frac{1}{2}(1-\alpha)x_2^TAx_2+\alpha b^Tx_1+(1-\alpha)b^Tx_2$$

אגף שמאל של המשוואה הוא

$$f(\alpha x_1+(1-\alpha)x_2)=\frac{1}{2}\left[\alpha x_1+(1-\alpha)x_2\right]^T A\left[\alpha x_1+(1-\alpha)x_2\right]+b^T\left[\alpha x_1+(1-\alpha)x_2\right]=$$
$$=\frac{1}{2}\alpha^2x_1^TAx_1+\frac{1}{2}(1-\alpha)^2x_2^TAx_2+\alpha(1-\alpha)x_1^TAx_2+\alpha b^Tx_1+(1-\alpha)b^Tx_2$$

נבדוק מתי הביטוי הבא הוא אי-שלילי

$$\alpha f(x_1)+(1-\alpha)f(x_2)-f(\alpha x_1+(1-\alpha)x_2)=$$
$$=\frac{1}{2}\alpha x_1^TAx_1+\frac{1}{2}(1-\alpha)x_2^TAx_2+\alpha b^Tx_1+(1-\alpha)b^Tx_2-$$

$$\left(\frac{1}{2}\alpha^2x_1^TAx_1+\frac{1}{2}(1-\alpha)^2x_2^TAx_2+\alpha(1-\alpha)x_1^TAx_2+\alpha b^Tx_1+(1-\alpha)b^Tx_2\right)=$$
$$=\frac{1}{2}\alpha x_1^TAx_1+\frac{1}{2}(1-\alpha)x_2^TAx_2-\frac{1}{2}\alpha^2x_1^TAx_1-\frac{1}{2}(1-\alpha)^2x_2^TAx_2-\alpha(1-\alpha)x_1^TAx_2=$$
$$=\frac{1}{2}\left\{\left(\alpha-\alpha^2\right)x_1^TAx_1+\left[(1-\alpha)-(1-\alpha)^2\right]x_2^TAx_2-2\alpha(1-\alpha)x_1^TAx_2\right\}=$$
$$=\frac{1}{2}\left\{\alpha(1-\alpha)x_1^TAx_1+(1-\alpha)\left[1-(1-\alpha)\right]x_2^TAx_2-2\alpha(1-\alpha)x_1^TAx_2\right\}=$$

$$=\frac{1}{2}\alpha(1-\alpha)\left(x_1^T Ax_1 + x_2^T Ax_2 - 2x_1^T Ax_2\right)=$$
$$=\frac{1}{2}\alpha(1-\alpha)\left(x_1-x_2\right)^T A\left(x_1-x_2\right)\geq 0$$

מאחר ו- $\alpha,(1-\alpha)\geq 0$ אז מספיק לבדוק מתי מתקיים האי"ש

$$\forall x_1,x_2\in\mathbb{R}^n:\left(x_1-x_2\right)^T A\left(x_1-x_2\right)\geq 0 \quad (1)$$

נגדיר $\tilde{x}=x_1-x_2$. מאחר וצריך לבדוק את אי"ש (1) לכל $\forall x_1,x_2\in\mathbb{R}^n$ אז זה שקול לבדוק את האי"ש

$$\forall\tilde{x}:\ \tilde{x}^T A\tilde{x}\geq 0 \quad (2)$$

נשים לב כי הביטוי באי"ש (2) הוא התבנית הריבועית של המטריצה A. לכן אי"ש (2) מתקיים כאשר $A\succeq 0$.

לסיכום, הפונקציה הריבועית קמורה אם המטריצה A היא מטריצה חיובית חצי מוגדרת.

ב).
נניח כי $f(x)\in C^1$. הגרדיאנט של הפונקציה הוא

$\nabla f(x)=Ax+b$.

נבדוק תחת איזה תנאים מתקיים האי"ש:

$$\forall x,x_0\in\mathbb{R}^n:\ f(x)\geq f(x_0)+\nabla f(x_0)^T\left(x-x_0\right)$$

נציב:

$$\frac{1}{2}x^T Ax+b^T x\geq\frac{1}{2}x_0^T Ax_0+b^T x_0+\left(x-x_0\right)^T\nabla f(x_0)$$
$$\frac{1}{2}x^T Ax+b^T x-\frac{1}{2}x_0^T Ax_0-b^T x_0-\left(x-x_0\right)^T(Ax_0+b)\geq 0$$

$$\frac{1}{2}x^T Ax+b^T x-\frac{1}{2}x_0^T Ax_0-b^T x_0-x^T Ax_0-b^T x+x_0^T Ax_0+b^T x_0\geq 0$$

$$\frac{1}{2}\left(x^T Ax+x_0^T Ax_0-2x^T Ax_0\right)\geq 0$$

$$\frac{1}{2}\left(x-x_0\right)^T A\left(x-x_0\right)\geq 0 \quad (3)$$

נגדיר $\tilde{x}=x-x_0$. באופן זהה לסעיף הקודם נקבל כי אי"ש (3) מתקיים כאשר $A\succeq 0$.

ג). נניח כי $f(x)\in C^2$. ההסיאן של הפונקציה הוא

$\nabla^2 f(x)=A$

הפונקציה היא קמורה אם

$\nabla^2 f(x)=A\succeq 0$

When the objective function is convex, local and global minimizers are simple to characterize.

Theorem 1. *When f is convex, any local minimizer $\mathbf{x}^*$ is a global minimizer of f. If in addition f is differentiable, then any stationary point $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*) = 0$ is a global minimizer of f.*

We will only consider the proof of the first part of the Theorem here:

PROOF. Suppose that x^* is a local but not a global minimizer. Then we can find a point $z \in \mathbb{R}^n$ with $f(z) < f(x^*)$. Consider the line segment that joins x^* to z, that is,

$$x = \lambda z + (1 - \lambda)x^*, \qquad \text{for some } \lambda \in (0, 1]. \tag{2.7}$$

By the convexity property for f, we have

$$f(x) \leq \lambda f(z) + (1 - \lambda) f(x^*) < f(x^*). \tag{2.8}$$

Any neighborhood $\mathcal{N}$ of x^* contains a piece of the line segment (2.7), so there will always be points $x \in \mathcal{N}$ at which (2.8) is satisfied. Hence, x^* is not a local minimizer.

The proof of the second part of the Theorem is obtained by contradiction.

These results provide the foundation of unconstrained optimization algorithms. If we have a convex function, then every local minimizer is also a global minimizer, and hence if we have methods that reach local minimizers, we can reach the global minimizer. This is why convex problems are very popular in science, although in some cases there is no choice but to solve non-convex problems. In all algorithms we seek a point $\mathbf{x}^*$ where $\nabla f(\mathbf{x}^*) = 0$.

2 Iterative methods for unconstrained optimization

Just like linear systems, optimization can be carried out by iterative methods. Actually, in optimization, it is usually not possible to directly find a minimum of a function, and so iterative methods are much more essential in optimization than in numerical linear algebra. On the other hand, we have learned about minimizing quadratic functions, which is equivalent to solving linear systems. Since every function is locally approximated by a quadratic function (and specifically near its minimum) everything we learned about iterative methods for linear systems can be useful in the context of optimization.

2.1 Steepest descent (SD)

Earlier, we met the steepest descent method

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

which we analysed for positive definite linear systems $A\mathbf{x} = \mathbf{b}$. In fact, we saw that if we look at the equivalent quadratic minimization, then the matrix A has to be positive definite, which means that the quadratic function f has to be convex (this does not mean that a general f has to be convex for Steepest Descent to work). Just like quadratic (algebraic) case, $\alpha^{(k)}$ can be chosen sufficiently small, whether a constant $\alpha^{(k)} = \alpha$ or not, so that the method converges to a local minimum. However, a more common way is to look at $\alpha^{(k)}$ as a step size, and choose it in some wise way.

2.2 A descent direction

In the method of SD, we chose the direction

$$\mathbf{d}_{SD} = -\nabla f(\mathbf{x}^{(k)}), \tag{2}$$

as the most obvious choice for a descent direction (which requires a suitable step-size). Among all the directions that we could choose from $\mathbf{x}^{(k)}$, this is the one in which f decreases most rapidly. To verify this claim, let us first define a descent direction: a direction in which the value of f decreases. To define such a direction we use Taylor's theorem. Let $\mathbf{d}$ be a

direction of norm 1 ($\|\mathbf{d}\| = 1$), and let $\alpha > 0$ be a positive step size. We have:

$$f(\mathbf{x}^{(k)} + \alpha\mathbf{d}) = f(\mathbf{x}^{(k)}) + \alpha\langle\nabla f(\mathbf{x}^{(k)}), \mathbf{d}\rangle + \frac{1}{2}\alpha^2\langle\mathbf{d}, \nabla^2 f(\mathbf{x}^{(k)} + t\mathbf{d})\mathbf{d}\rangle, \quad t \in (0, \alpha).$$

Now, for $\alpha > 0$ *sufficiently small*, we will have that

$$f(\mathbf{x}^{(k)} + \alpha\mathbf{d}) < f(\mathbf{x}^{(k)}) \quad \Leftrightarrow \quad \langle\nabla f(\mathbf{x}^{(k)}), \mathbf{d}\rangle < 0, \tag{3}$$

which is the condition for $\mathbf{d}$ to qualify as a **descent direction**.

Corollary 1. *Any direction $\mathbf{d} = -M\nabla f$ such that $M \succ 0$ is a descent direction.*

It is easy to see that since we chose $\|\mathbf{d}\| = 1$ we have

$$\langle\nabla f, \mathbf{d}\rangle = \|\nabla f\|\|\mathbf{d}\|\cos(\theta) = \|\nabla f\|\cos(\theta),$$

where θ is the angle between the vectors ∇f and $\mathbf{d}$. to maximize the decrease in f we should choose $\mathbf{d}$ such that $\cos(\theta) = -1$, which will be satisfied in $\theta = 180°$. That is the opposite direction from the gradient, or, $-\nabla f$. This is the reason why (2) qualifies as the "steepest descent" direction.

Interpretation of SD as quadratic minimization We will look at more general methods next, but first observe that steepest descent method (with a pre-defined step size α) can be viewed as a step that minimizes the quadratic function that approximates $f(\mathbf{x})$ around $\mathbf{x}^{(k)}$:

$$\mathbf{d}_{SD}^{(k)} = \arg\min_{\mathbf{d}} \left\{ f(\mathbf{x}) + \langle\nabla f(\mathbf{x}^{(k)}), \mathbf{d}\rangle + \frac{1}{2\alpha}\langle\mathbf{d}, \mathbf{d}\rangle \right\} = -\alpha\nabla f(\mathbf{x}^{(k)}).$$

Then:

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

This derivation assumes that α is either constant or known a-priori. Equivalently, one can look at this with having $\alpha = 1$ for the derivation, and then having a linesearch procedure for choosing α (given later). But here, we see that in fact, the step-size is related to the approximation of the Hessian as $\nabla^2 f(\mathbf{x}^k) \approx \frac{1}{\alpha}I$. We will now see some other methods that follow the same pattern, and later see a method for determining the step length α through linesearch.

2.3 Newton, quasi-Newton

One of the most important methods outside of steepest descent is Newton's method, which is much more powerful than SD, but also may be more expensive. Recall again the Taylor expansion

$$f(\mathbf{x}^{(k)} + \boldsymbol{\varepsilon}) = f(\mathbf{x}^{(k)}) + \langle \nabla f(\mathbf{x}^{(k)}), \boldsymbol{\varepsilon} \rangle + \frac{1}{2}\langle \boldsymbol{\varepsilon}, \nabla^2 f(\mathbf{x}^{(k)})\boldsymbol{\varepsilon} \rangle + O(\|\boldsymbol{\varepsilon}\|^3). \tag{4}$$

Now we choose the next step $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\varepsilon}$ by a minimizing Eq. (4) without the $O(\|\boldsymbol{\varepsilon}\|^3)$ term with respect to $\boldsymbol{\varepsilon}$. That is, the Newton direction $\mathbf{d}_N$ is chosen by the following quadratic minimization:

$$\mathbf{d}_N^{(k)} = \arg\min_{\mathbf{d}} \left\{ f(\mathbf{x}) + \langle \nabla f(\mathbf{x}^{(k)}), \mathbf{d} \rangle + \frac{1}{2}\langle \mathbf{d}, \nabla^2 f(\mathbf{x}^{(k)})\mathbf{d} \rangle \right\} = -(\nabla^f(\mathbf{x}^{(k)})^{-1})\nabla f(\mathbf{x}^{(k)}).$$

This minimization is similar to the one mentioned in SD, but now it has a $\nabla^2 f(\mathbf{x}^{(k)})$ instead of a matrix $\frac{1}{\alpha}I$. This shows us that in SD we essentially approximate the real Hessian with a scaled identity matrix.

The Newton procedure leads to a search direction

$$\mathbf{d}_N = -(\nabla^2 f(\mathbf{x}^{(k)}))^{-1}\nabla f(\mathbf{x}^{(k)}),$$

which is the Newton's direction. This is the best search direction that is practically used, and it is relatively hard to obtain: first, the function f should be twice differentiable. Second, one has to compute the Hessian matrix or at least know how to solve a non-trivial linear system involving the Hessian matrix. This has to be computed at *each iteration*, which may be costly. Solving the linear system can be done directly through an LU decomposition, or iteratively using an appropriate method. If we choose a constant steplength, then one can show that the best step-length for Newton's method is asymptotically 1.0. However, throughout the iterations it is better to perform a lineaserach over the direction $\mathbf{d}_N$ to guarantee the decreasing values of $f(\mathbf{x}^{(k)})$ and convergence.

It is important to note that the Newton's method should be used with care. There is no guarantee that $\mathbf{d}$ is a descent direction, or that the matrix $(\nabla^2 f(\mathbf{x}^{(k)}))$ is invertible. If the problem is convex and the Hessian is positive definite - we have nothing to worry about.

Quasi Newton methods Applying the Newton's method is expensive. In Quasi-Newton we approximate the Hessian by some matrix $M(\mathbf{x}^{(k)})$, and at each step minimize

$$\mathbf{d}_{QN}^{(k)} = \arg\min_{\mathbf{d}} \left\{ f(\mathbf{x}) + \langle \nabla f(\mathbf{x}^{(k)}), \mathbf{d} \rangle + \frac{1}{2} \langle \mathbf{d}, M(\mathbf{x}^{(k)})\mathbf{d} \rangle \right\} = -(M(\mathbf{x}^{(k)})^{-1}) \nabla f(\mathbf{x}^{(k)}). \quad (5)$$

M can be chosen as diagonal, $M = cI$ or $M = \mathrm{diag}(\nabla^2 f)$, which is easily invertible, or by other ways. We will not go into this further in this course, but the limited memory Broyden—Fletcher-–Goldfarb-–Shanno (LBFGS) method is one of the most popular quasi-Newton methods in the literature. It iteratively builds a low rank approximation of the Hessian from previous search directions, which is somewhat similar to some Krylov methods that we've seen earlier in this course (e.g. GMRES).

All the methods above are one-point iterations, and are summarized in Algorithm 1.

Algorithm: General one-point iterative method
 # Input: Objective: $f(\mathbf{x})$ to minimize.
for $k = 1, ..., maxIter$ **do**
  Compute the gradient: $\nabla f(\mathbf{x}^k)$.
  Define the search direction $\mathbf{d}^{(k)}$, by essentially solving a quadratic minimization.
  Choose a step-length $\alpha^{(k)}$ (possibly by linesearch)
  Apply a step:
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k+1)} + \alpha^{(k)} \mathbf{d}^{(k)}.$$
  if $\frac{\|\nabla f(\mathbf{x}^{(k+1)})\|}{\|\nabla f(\mathbf{x}^{(1)})\|} < \epsilon$ *or alternatively* $\frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^{(k)}\|} < \epsilon$. **then**
    Convergence is reached, stop the iterations.
  end
end
Return $\mathbf{x}^{(k+1)}$ as the solution.

Algorithm 1: General one-point iterative method for unconstrained optimization

2.4 Line-search methods

Line-search methods are among the most common methods in optimization. Assume we are at iteration k, at point $\mathbf{x}^{(k)}$. Assume that we have a descent direction $\mathbf{d}^{(k)}$ in which the

function decreases. Line-search methods performs

$$\mathbf{x}^{(k+1)} = x^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)} \tag{6}$$

and choose $\alpha^{(k)}$ such that $f(x^{(k)} + \alpha\mathbf{d}^{(k)})$ is exactly or approximately minimized for α. In the quadratic SD case, we had a closed term that minimizes this linesearch.

When computing the step length $\alpha^{(k)}$, we are essentially minimizing a one-dimensional function

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

with respect to α. All 1-D minimization methods can be used here, but we face a tradeoff. We would like to choose $\alpha^{(k)}$ to substantially reduce f , but at the same time we do not want to spend too much time making the choice. In general, it is too expensive to identify a minimizer, as it may require too many evaluations of the objective f. We assume here that the computation of $f(\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)})$ is just as expensive as computing $f(\mathbf{w})$ for some arbitrary vector $\mathbf{w}$. If this is not the case, we may be able to allow ourselves to invest more iterations in minimizing $\phi(\alpha)$. However, note that minimizing ϕ is only locally optimal (a greedy choice), and there is no guarantee that an exact linesearch minimization is indeed the right choice for the fastest convergence, not to mention the cost of doing that.

More practical strategies perform an inexact line search to identify a step length that achieves adequate reductions in f at minimal cost. Typical line search algorithms try out a sequence of candidate values for α, stopping to accept one of these values when certain conditions are satisfied. Sophisticated line search algorithms can be quite complicated. We now show a practical termination condition for the line search algorithm. We will later show, using a simple quadratic example, that effective step lengths need not lie near the minimizers of $\phi(\alpha)$.

Backtracking line search using the Armijo condition We will now show one of the most common line search algorithms, called "backtracking" line search, also known as the Armijo rule. It involves starting with a relatively large estimate of the step size α, and iteratively shrinking the step size (i.e., "backtracking") until a sufficient decrease of the objective function is observed. The motivation is to choose $\alpha^{(k)}$ to be as large as possible while having a sufficient decrease of the objective at the same time. We are *not* minimizing

ϕ. This is done by choosing α_0 rather large, and choosing a decrease factor $0 < \beta < 1$. We will examine the values of $\phi(\alpha_j)$ for decreasing values $\alpha_j = \beta^j \alpha_0$ (using $j = 0, 1, 2, ...$) until the following condition is satisfied:

$$f(\mathbf{x}^{(k)} + \alpha_j \mathbf{d}^{(k)}) \leq f(\mathbf{x}^{(k)}) + c\alpha_j \langle \nabla f, \mathbf{d}^{(k)} \rangle. \tag{7}$$

$0 < c < 1$ is usually chosen small (about 10^{-4}), and β is usually chosen to be about 0.5. We are guaranteed that such a point exists, because we assume that $\mathbf{d}^{(k)}$ is a descent direction. That is, we know that $\langle \nabla f, \mathbf{d}^{(k)} \rangle < 0$, and using the Taylor theorem we have

$$f(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}) = f(\mathbf{x}^{(k)}) + \alpha \langle \nabla f, \mathbf{d}^{(k)} \rangle + O(\alpha^2 \|\mathbf{d}^{(k)}\|^2).$$

It is clear that for some α small enough we will have

$$f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}) = -\alpha \langle \nabla f, \mathbf{d}^{(k)} \rangle + O(\alpha^2 \|\mathbf{d}^{(k)}\|^2) > 0.$$

In our stopping rule we usually set c to be small, so we choose α to be relatively large. Algorithm 2 summarizes the backtracking linesearch procedure. Common "upgrade" for this procedure are

- Choose α_0 for iteration k as $\frac{1}{\beta}\alpha^{(k-1)}$ or $\frac{1}{\beta^2}\alpha^{(k-1)}$.
- Instead of only reducing the α_j's, also try to enlarge them by $\alpha_{j+1} = \frac{1}{\beta}\alpha_j$ as long as the Armijo condition is satisfied.

Example 4 (Newton vs. Steepest descent)**.** *We will now consider the minimization of the Rosenbrock "banana" function $f(\mathbf{x}) = (a - x_1)^2 + b(x_2 - x_1^2)^2$, where a, b are parameters that determine the "difficulty" of the problem. Here we choose $b = 5$, and $a = 1$. The minimum of this function is at $[a, a^2]$, where $f = 0$ (in our case it's the point [1,1]).*

The gradient and Hessian of this function are

$$\nabla f(x) = \begin{bmatrix} -4bx_1(x_2 - x_1^2) - 2(a - x_1) \\ 2b(x_2 - x_1^2) \end{bmatrix} \quad \nabla^2 f(x) = \begin{bmatrix} -4b(x_2 - x_1^2) + 8bx_1^2 + 2 & -4bx_1 \\ -4bx_2 & 2b \end{bmatrix}$$

In the code below we apply the SD and Newton's algorithms starting from the point $[-1.4, 2]$.

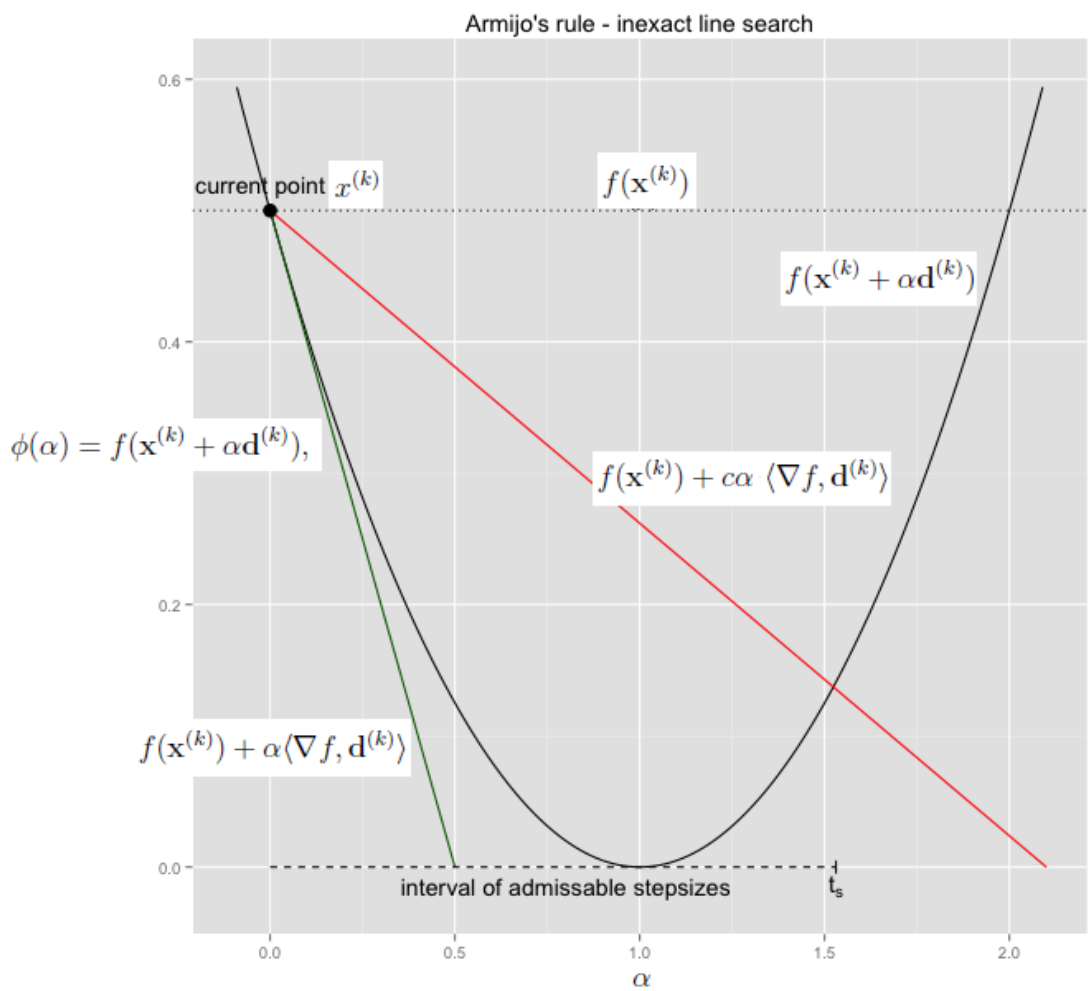

Figure 5: Armijo rule

Algorithm: Armijo Linesearch
 # Input: Iterate $\mathbf{x}$*, objective* $f(\mathbf{x})$*, gradient* $\nabla f(x)$*, descent direction* $\mathbf{d}$*.*
 # Constants: $\alpha_0 > 0$*,* $0 < \beta < 1$*,* $0 < c < 1$*.*
for $j = 0, ..., maxIter$ **do**
  Compute the objective $\phi(\alpha_j) = f(\mathbf{x} + \alpha_j \mathbf{d})$.
  if $f(\mathbf{x} + \alpha_j \mathbf{d}) \leq f(\mathbf{x}^{(k)}) + c\alpha_j \langle \nabla f, \mathbf{d}^{(k)} \rangle$. **then**
    Return $\alpha = \alpha_j$ as the chosen step-length.
  else
    $\alpha_{j+1} = \beta \alpha_j$
  end
end
 # If maxIter iterations were reached, α *is too small and we are probably stuck. # In this case terminate the minimization.*

Algorithm 2: Armijo backtracking linesearch procedure.

After 100 iterations, SD reached $[0.957531, 0.915136]$ *on his way to* $[1, 1]$*. The Newton method solved the problem in only 9 iterations.*

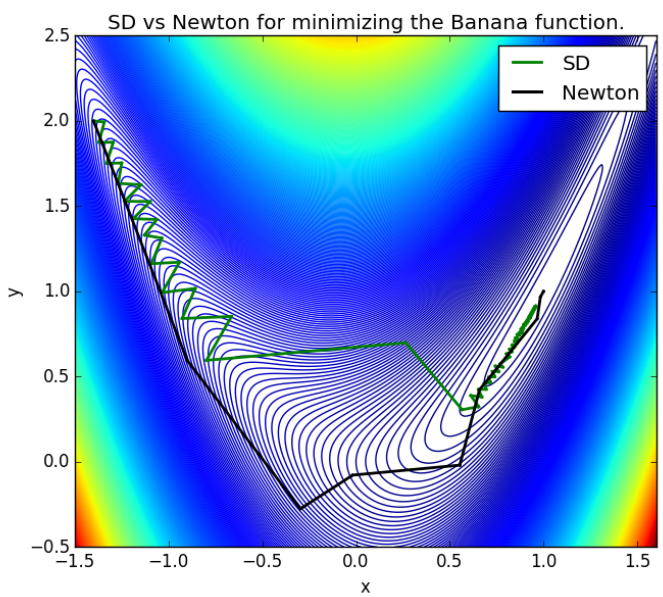

Figure 6: The convergence path of steepest descent and Newton's method for minimizing the "banana" function.

```
using PyPlot;close("all");
xx = -1.5:0.01:1.6
yy= -0.5:0.01:2.5;
X = repmat(xx ,1,length(yy))';
Y = repmat(yy ,1,length(xx));
a = 1; b = 5;
F = (a-X).^2 + b*(Y-X.^2).^2
figure(); contour(X,Y,F,500); #hold on; axis image;
xlabel("x"); ylabel("y"); title("SD vs Newton for minimizing the Banana function.")
f = (x)->((a-x[1]).^2 + b*(x[2]-x[1].^2).^2)
g = (x)->[-4*b*x[1]*(x[2]-x[1]^2)-2*(a-x[1]); 2*b*(x[2]-x[1]^2)];
H = (x)->[-4b*(x[2]-x[1]^2)+8*b*x[1]^2+2 -4*b*x[1] ; -4*b*x[1] 2*b];
## Armijo parameters:
alpha0 = 1.0; beta = 0.5; c = 1e-4;
function linesearch(f::Function,x,d,gk,alpha0,beta,c)
alphaj = alpha0;
for jj = 1:10
    x_temp = x + alphaj*d;
    if f(x_temp) <= f(x) + alphaj*c*dot(d,gk)
        break;
    else
        alphaj = alphaj*beta;
    end
end
return alphaj;
end
println("*********** SD iterations *****************")
x_SD = [-1.4;2.0]; # initial guess
## SD Iterations
for k=1:100
    gk = g(x_SD);
    d_SD = -gk;
    x_prev = copy(x_SD);
    alpha_SD = linesearch(f,x_SD,d_SD,gk,0.25,beta,c);
    x_SD=x_SD+alpha_SD*d_SD;
    plot([x_SD[1];x_prev[1]],[x_SD[2];x_prev[2]],"g",linewidth=2.0); println(x_SD);
end;
println("*********** Newton iterations *****************")
x_N = [-1.4;2.0]; # initial guess
for k=1:10
    gk = g(x_N);
    Hk = H(x_N);
    d_N = -(Hk)\gk;
    x_prev = copy(x_N);
    alpha_N  = linesearch(f,x_N,d_N,gk,alpha0,beta,c);
    x_N=x_N+alpha_N*d_N;
    plot([x_N[1];x_prev[1]],[x_N[2];x_prev[2]],"k",linewidth=2.0); println(x_N);
end;
legend(("SD","Newton"))
```

2.5 Coordinate descent methods

Coordinate descent methods are similar in principle to the Gauss-Seidel method for linear systems. When we minimize $f(\mathbf{x})$, we iterate over all entries i of $\mathbf{x}$, and change each scalar entry x_i so that the i-th $f(\mathbf{x})$ is minimized with respect to x_i given that the rest of the variables are fixed.

Algorithm: Coordinate Descent
 # Input: Objective: $f(\mathbf{x})$ to minimize.
for $k = 1, ..., maxIter$ **do**
  #sweep through all scalar variables x_i and (approximately) minimize $f(\mathbf{x})$,
  # according x_i in turn, assuming the rest of the variables are fixed.
  for $i = 1, ..., n$ **do**
    $x_i^{(k+1)} \leftarrow \arg\min_{x_i} f(x_1^{(k+1)}, x_2^{(k+1)}, ..., x_{i-1}^{(k+1)}, x_i, x_{i+1}^{(k)}, ..., x_n^{(k)})$
  end
  # Check if convergence is reached.
end
Return $\mathbf{x}^{(k+1)}$ as the solution.

Algorithm 3: Coordinate descent

By doing this, we essentially zero the i-th entry of the gradient, i.e., update x_i such that

$$\frac{\partial f}{\partial x_i} = 0.$$

Similarly to the variational property of Gauss-Seidel, the value of f is monotonically non-increasing with each update. If f is bounded from below, the series $\{f(\mathbf{x}^{(k)})\}$ converges. This does not guarantee convergence to a minimizer, but it is a good property to have. There are many rules in which order to choose the variables x_i. For example, lexicographic or random orders are common. One of the requirements to guarantee convergence is that all the variables will be visited periodically. That is, we cannot guarantee convergence if a few of the variables are not visited.

Coordinate descent methods are most common in non-smooth optimization, and especially effective when one-dimensional minimization is easy to apply.

Example 5 (Coordinate descent for the "banana" function)**.** *Assume again that we are minimizing $f(\mathbf{x}) = (a - x_1)^2 + b(x_2 - x_1^2)^2$, where a, b are parameters $b = 10$, and $a = 1$.*

We've seen that

$$\nabla f(x) = \begin{bmatrix} -4bx_1(x_2 - x_1^2) - 2(a - x_1) \\ 2b(x_2 - x_1^2) \end{bmatrix},$$

and in coordinate descent we're solving each of these equations in turn. We start with the second variable, because the updates regarding the second variable are easy to define (assuming $b \neq 0$*)*

$$\frac{\partial f}{\partial x_2} = 2b(x_2 - x_1^2) = 0 \Rightarrow x_2 = x_1^2.$$

For the first variable, we have

$$\frac{\partial f}{\partial x_1} = -4bx_1(x_2 - x_1^2) - 2(a - x_1) = 0 \Rightarrow x_1 = ?[\textit{No closed form solution}]$$

This time there is no closed form solution to the problem, and in fact there may be more than one minimum point here. A common option in such cases is to apply a one-dimensional Newton step for x_1*:*

$$x_1^{new} = x_1 - \alpha \frac{f'_{x_1}}{f''_{x_1}} = x_1 - \alpha \frac{-4bx_1(x_2 - x_1^2) - 2(a - x_1)}{-4b(x_2 - x_1^2) + 8bx_1^2 + 2}$$

where α *is obtained by linesearch with respect to minimizing* $f(x_1 + \alpha d_1, x_2)$ *and* d_1 *is the Newton search direction for* x_1*.*

The code for the CD updates appears next. It works in addition to the code in the previous examples. After 100 iterations, CD iterates reached $[0.997156, 0.99432]^\top$ *on their way to [1,1], which is significantly better than SD but worse than Newton's method.*

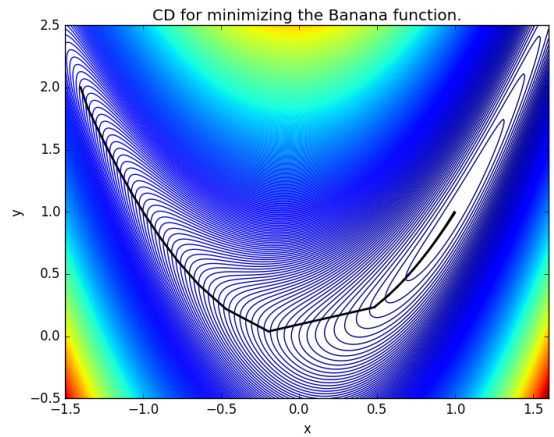

Figure 7: The convergence path of the coordinate descent method for minimizing the "banana" function.

```
## CD Iterations (this code comes in addition to the first section of code in the previous example)
x_CD = [-1.4;2.0];
println("*********** Newton *****************")
for k=1:100
    # first variable Newton update:
    gk = -4*b*x_CD[1]*(x_CD[2]-x_CD[1]^2)-2*(a-x_CD[1]); # This is g[1]
    Hk = -4*b*(x_CD[2]-x_CD[1]^2)+8*b*x_CD[1]^2+2;   # This is H[1,1]
    d_1 = -gk/Hk; ## Hk is a scalar
    f1 = (x)->((a-x).^2 + b*(x_CD[2]-x.^2).^2) # f1 accepts a scalar.
    x_prev = copy(x_CD);
    alpha_1  = linesearch(f1,x_CD[1],d_1,gk,alpha0,beta,c);
    x_CD[1] = x_CD[1]+alpha_1*d_1;
    # second variable update
    x_CD[2] = x_CD[1]^2;
    plot([x_CD[1];x_prev[1]],[x_CD[2];x_prev[2]],"k",linewidth=2.0); println(x_CD);
end;
```

2.6 Gauss-Newton and Lavenberg-Marquardt for non-linear least squares.

In many scenarios we wish to fit given data to a model using some parameters. We already know that Least Squares is a popular approach for penalizing for the model error. However, what if the model is non-linear? One of the most popular approaches to solve this is the Gauss-Newton method.

The typical scenario: data fitting. Suppose that you are given with (possibly noisy) observations $\mathbf{y}^{obs}$ and you wish to fit a model $\mathbf{f}(\theta)$ to this data, for prediction or analysis. θ is a vector of model parameters. A popular choice would be to solve

$$\min_{\theta} F(\theta) = \min_{\theta} \frac{1}{2}\|\mathbf{f}(\theta) - \mathbf{y}^{obs}\|_2^2 \tag{8}$$

which may be accompanied with some regularization over θ, which we will ignore here.

A popular approach to solve the problem above is by some descent algorithm. For example the steepest descent method would yield:

$$\theta^{(k+1)} = \theta^{(k)} - \alpha^{(k)} \nabla_\theta F(\theta^{(k)})$$

This method, however, often converges slowly, and using a higher-order method is beneficial. The method of Gauss-Newton states to approximate $\mathbf{f}$ up to first order

$$\mathbf{f}(\theta^{(k)} + \mathbf{d}) \approx \mathbf{f}(\theta^{(k)}) + \mathbf{J}(\theta^{(k)})\mathbf{d}$$

where $\mathbf{J}(\theta^{(k)})$ is the Jacobian of $\mathbf{f}$ w.r.t θ, and $\mathbf{d}$ is a perturbation in θ. Using this approximation, we find the search direction by minimizing the resulting approximation:

$$\mathbf{d}_{GN}^{(k)} = \arg\min_{\mathbf{d}} \frac{1}{2}\|\mathbf{f}(\theta^{(k)}) + \mathbf{J}(\theta^{(k)})\mathbf{d} - \mathbf{y}^{obs}\|_2^2 = -(\mathbf{J}^\top \mathbf{J})^{-1} \nabla F(\theta^{(k)}), \tag{9}$$

where $\nabla F(\theta) = \mathbf{J}^\top(\mathbf{f}(\theta) - \mathbf{y}^{obs})$. The stepsize is obtained by linesearch (e.g., Armijo):

$$\theta^{(k+1)} = \theta^{(k)} + \alpha^{(k)} \mathbf{d}_{GN}^{(k)}.$$

In some cases, the Jacobian $\mathbf{J}$ may be ill-conditioned or of low-rank. The reason is that

in some cases, the Jacobian is defined by the data itself which is an input to the solver. In such cases, the system involving $\mathbf{J}^\top\mathbf{J}$ in the previous section becomes unstable. A quick fix for this results in the **Levenberg–Marquardt** variant of GN:

$$\mathbf{d}_{LM}^{(k)} = \arg\min_{\mathbf{d}} \frac{1}{2}\|\mathbf{f}(\theta^{(k)}) + \mathbf{J}(\theta^{(k)})\mathbf{d} - \mathbf{y}^{obs}\|_2^2 + \frac{\mu}{2}\|\mathbf{d}\|_2^2, \tag{10}$$

for some parameter $\mu > 0$. This will lead to the same descent direction as in GN, but with the matrix $\mathbf{J}^\top\mathbf{J} + \mu\mathbf{I}$ instead of $\mathbf{J}^\top\mathbf{J}$. This stabilizes the GN algorithm, but also requires a choice for a modest μ (reminder: μ should be slightly higher than the small eigenvalues of $\mathbf{J}^\top\mathbf{J}$, which indeed can be a bit tricky to choose).

2.7 Iterative Re-weighted Least Squares (IRLS)

There are many cases in real life applications where we wish to solve problems that we don't really know how to handle efficiently. For example, it includes a different loss than ℓ_2, over the data fit term. One such common example is a minimization of a linear equation in ℓ_p for $p \neq 2$:

$$\arg\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_p^p. \tag{11}$$

The method of "Iterated Re-weighted Least Squares" (IRLS) states that instead of solving a problem like (11) directly.

Suppose we wish to minimize

$$\mathbf{x}^* = \arg\min_x \phi(A\mathbf{x} - \mathbf{b}) = \arg\min_x \sum_i \phi(\mathbf{a}_i^\top\mathbf{x} - b_i)$$

where ϕ is some scalar loss function that operates on every entry and is summed. In IRLS, we replace this with a weighted sum of squares

$$\mathbf{x}^* = \arg\min_x \sum_i w_i^\phi(\mathbf{a}_i^\top\mathbf{x} - b_i)^2,$$

with appropriate weights.

The Algorithm (IRLS) states: fix the weights for current iteration:

$$\mathbf{x}^{(k+1)} = \arg\min_x \sum_i w_i^{(k)} (\mathbf{a}_i^\top \mathbf{x}^{(k)} - b_i)^2 = \arg\min_x \|A\mathbf{x} - \mathbf{b}\|_{W^{(k)}}^2, \tag{12}$$

with the same appropriate weights, but now, $w_i^{(k)}$ are constants for the (k)-th iteration. We therefore iteratively solve the original problem by solving a series of (diagonally) weighted LS problems. Note that $W^{(k)}$ depends on $\mathbf{x}^{(k)}$, but is kept fixed in the minimization to allow us to have a closed form solution. That is the reason that a few iterations of IRLS are needed.

Example 6 (ℓ_1-norm minimization using IRLS)**.** *Suppose we wish to solve*

$$\arg\min_x \|A\mathbf{x} - \mathbf{b}\|_1.$$

The objective function is a non-smooth function, and has no closed form solution in this general case. Instead of solving the problem directly, we define

$$w_i^{(k)} = \frac{1}{|\mathbf{a}_i^T \mathbf{x}^{(k)} - \mathbf{b}| + \epsilon},$$

where $\epsilon > 0$ is small, and iteratively solve

$$\mathbf{x}^{(k+1)} = \arg\min_x \frac{1}{2} \|A\mathbf{x} - \mathbf{b}\|_{W^{(k)}}^2.$$

Note that the factor $1/2$ is in fact subtle, but now, here, has no meaning since the argmin is identical with or without it. Note that if we set $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$, then

$$\|\mathbf{r}^{(k)}\|_1 = (\mathbf{r}^{(k)})^\top W^{(k)} \mathbf{r}^{(k)}.$$

Example 7. *ℓ_p-norm minimization Suppose that we wish to solve*

$$\arg\min_x \|A\mathbf{x} - \mathbf{b}\|_p.$$

Here, the smoothness of the objective depends on p, and the problem is convex only for $p \geq 1$. There's no closed form solution except $p = 2$. Instead of solving the problem directly,

we define $w_i^{(k)} = (|\mathbf{a}_i^T\mathbf{x}^k - \mathbf{b}| + \epsilon)^{p-2}$*, where* $\epsilon > 0$ *is small. Then, we iteratively solve*

$$\mathbf{x}^{(k+1)} = \arg\min_x \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|^2_{W^{(k)}}.$$

Again, the $1/2$ *is in fact subtle, but now, here, has no meaning. See more here:* `https://cnx.org/contents/krkDdys0@12/Iterative-Reweighted-Least-Squares`

IRLS—a more general description We will now ignore the inner structure of the objective and only relate to it as $\mathbf{r}$. Suppose we wish to minimize $f(\mathbf{r})$ of the form

$$\mathbf{r}^* = \arg\min_r \sum_i \phi(r_i)$$

Algorithm (IRLS): fix the weights for current iteration:

$$\mathbf{r}^{(k+1)} = \arg\min_r \sum_i w_i^{(k)}(r_i)^2 = \arg\min_r \|\mathbf{r}\|^2_{W^{(k)}},$$

with appropriate weights. Denote the quadratic approximation

$$\tilde{f}_k(\mathbf{r}) = \|\mathbf{r}\|^2_{W^{(k)}}$$

The minimum of the objective is defined by $\nabla f = 0$, and that's the property that we wish to replicate in our approximation, so the minimum is obtained at the same point $\mathbf{x}$. For that, we wish that the grad of the two functions is equal between the two versions:

$$\nabla f(\mathbf{r}) = \nabla \tilde{f}_k(\mathbf{r}).$$

So, we get:

$$(\nabla \tilde{f}_k(\mathbf{r}))_i = 2w_i^{(k)} r_i$$

$$(\nabla f(x))_i = \phi'(r_i)$$

Hence, we can set: $w_i^{(k)} = \frac{\phi'(r_i)}{2r_i}$. For ℓ_1 we will get: $w_i^{(k)} = \frac{sign(r_i)}{2r_i} = \frac{1}{2|r_i|}$, and here's where the $1/2$ comes from in the objective in the example above.

This derivation is important when we minimize sums of functions:

$$min_x(f_1(\mathbf{x}) + f_2(\mathbf{x})),$$

which is very common—one of these terms is usually a data-fit term, and the other is a regularization term.

2.8 Non-linear Conjugate Gradient methods

Earlier we've seen the Conjugate Gradient methods for linear systems. This method has a lot of nice properties and advantages. It turns out that the linear CG method can be extended to non-linear unconstrained optimization. This extension has many variants, and all of them reduce to the linear CG method when the function is quadratic. We will not study these methods in this course, but they are quite efficient and worthy of consideration. See `https://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient_method` for an intuitive explanation.

2.9 Inexact Newton Methods - Newton-PCG

In this approach we can utilize the power of the Newton method, if the dimension of the problem is too large for an exact inversion of the Hessian matrix. There are other reasons why it may not be possible or efficient, but they are usually application specific. In this method, instead of solving the system

$$\nabla^2 f(\mathbf{x}^{(k)})\mathbf{d} = -\nabla f(\mathbf{x}^{(k)})$$

directly, we apply an iterative method to solve it approximately. The common choice is Conjugate Gradients, either with or without a preconditioner M. This way, the nice properties of the linear CG method are in place and the solution is rather fast. This method is suitable for ill-conditioned problems, where a Hessian-vector product (that is necessary in CG) is easy to apply. This method is usually better than Steepest Descent (if it is applicable), becasue (1) the gradient has to be computed only a few times, and (2) the good properties of linear CG. The method is often comparable in performance to non-linear CG, but is more stable and understandable.

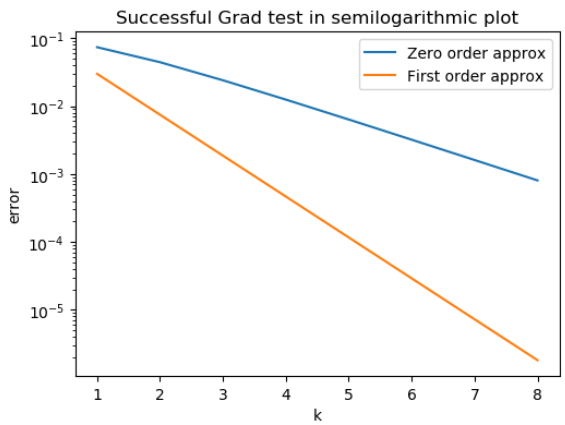

Figure 8: One way to present a successful gradient test. Since the scale is logarithmic in y, this is how the functions 0.5^k and 0.25^k (up to constant factors) would behave. The graphs are linear with different slopes. Code for generating the test is below.

2.10 Gradient and Jacobian verification

Each time we compute a non-linear function and its derivative, we need to make sure that our derivations are correct. It is better not to test this though the whole optimization process of our system, and do it part by part. This way, (1) we will be able to identify the errors in the right places, and (2) we won't be confused in the case where our optimization algorithm really does not supposed to work (the lack of minimization is a "feature instead of a bug").

2.10.1 The gradient test:

The simplest case is achieved for the gradient of a scalar function. Let $\mathbf{d}$ be a random vector, such that $\|\mathbf{d}\| = O(1)$. Then, we know that

$$f(\mathbf{x} + \epsilon\mathbf{d}) = f(\mathbf{x}) + \epsilon\mathbf{d}^T\nabla f + O(\epsilon^2)$$

Suppose that we have a code $\texttt{grad}(\mathbf{x})$ for computing $\nabla f(\mathbf{x})$. Then, we can compare

$$|f(\mathbf{x} + \epsilon\mathbf{d}) - f(\mathbf{x})| = O(\epsilon) \tag{13}$$

versus

$$|f(\mathbf{x} + \epsilon\mathbf{d}) - f(\mathbf{x}) - \epsilon\mathbf{d}^T\texttt{grad}(\mathbf{x})| = O(\epsilon^2). \tag{14}$$

We will test these two values for decreasing values of ϵ. For example $\epsilon_i = (0.5)^i \epsilon_0$ for some ϵ_0. The values of (13) should decrease linearly (be smaller by a factor of two for each iterate of i), and the values of (14) should (if the code is right) decrease quadratically (be smaller by a factor of four for each iterate of i).

An example for the gradient test of $\frac{1}{2}\|\mathbf{x}\|_2^2$ is given in the following code

```
using PyPlot
using LinearAlgebra
using Random
F = x->0.5*dot(x,x)
g_F = x->x
n = 20;
x = randn(n);
d = randn(n);
epsilon = 0.1;
F0 = F(x);
g0 = g_F(x);
y0 = zeros(8); y1 = zeros(8);
println("k\terror order 1 \t\t error order 2");
for k=1:8
        epsk = epsilon*(0.5^k);
        Fk = F(x+epsk*d);
        F1 = F0 + epsk*dot(g0,d);
        y0[k] = abs(Fk-F0);
        y1[k] = abs(Fk-F1)
        println(k,"\t",abs(Fk-F0),"\t",abs(Fk-F1));
end
semilogy(collect(1:8),y0);
semilogy(collect(1:8),y1);
legend(("Zero order approx","First order approx"));
title("Successful Grad test in semilogarithmic plot");
xlabel("k");
ylabel("error");

--- Output ---
k       error order 1            error order 2
1       0.07356278528598992     0.029820749771815258
2       0.044236580085952326    0.007455187442950262
3       0.02398208690371284     0.0018637968607393418
4       0.012456992667040367    0.00046594921518483545
5       0.0063449836373159485   0.00011648730379754113
6       0.0032016136446078036   2.9121825948053015e-5
7       0.0016080872787913592   7.2804564865691646e-6
8       0.0008058637535164337   1.820114123418648e-6
```

2.10.2 The Jacobian test:

The same technique can be used to test the computation of the Jacobian. Here, we do not necessarily compute the matrix explicitly, but compute its multiplication with a vector. Assume that you have the code $\texttt{JacMV}(\mathbf{x}, \mathbf{v})$, that computes the multiplication of the Jacobian with a vector $\mathbf{v}$, i.e., $\frac{\partial f}{\partial \mathbf{x}}\mathbf{v}$. The derivative is computed at the point $\mathbf{x}$. Similarly to before, we will use a vector $\mathbf{v} = \epsilon\mathbf{d}$

$$\|f(\mathbf{x} + \epsilon\mathbf{d}) - f(\mathbf{x})\| = O(\epsilon)$$

versus

$$\|f(\mathbf{x} + \epsilon\mathbf{d}) - f(\mathbf{x}) - \texttt{JacMV}(\mathbf{x}, \epsilon\mathbf{d})\| = O(\epsilon^2).$$

The test is exactly the same procedure as the gradient test, and the plots and table should be similar to the scalar ones presented earlier.

2.10.3 The transpose test:

In this test we assume that we have a routine $\texttt{JacTMV}(\mathbf{x}, \mathbf{v})$ that computes the multiplication $\left(\frac{\partial f}{\partial \mathbf{x}}\right)^T \mathbf{v}$. We will assume that our $\texttt{JacMV}$ function is correct, and passed the Jacobian test. Then we will use the equality

$$\mathbf{u}^T\mathbf{J}\mathbf{v} = \mathbf{v}^T\mathbf{J}^T\mathbf{u},$$

which holds for any two random vectors $\mathbf{u}, \mathbf{v}$. To pass the test, we will verify that

$$|\mathbf{u}^T\texttt{JacMV}(\mathbf{x}, \mathbf{v}) - \mathbf{v}^T\texttt{JacTMV}(\mathbf{x}, \mathbf{u})| < \epsilon,$$

where ϵ is the machine precision.

2.10.4 The direct Jacobian transposed test

In the previous paragraphs, you saw how to test the Jacobian-vector product and using it, the transposed Jacobian-vector product. For first order methods (e.g., SD, SGD) we may need only the transposed Jacobian-vector product. To test it directly (given a vector function $f()$), we will define a new scalar function:

$$g(\mathbf{x}) = \langle f(\mathbf{x}), \mathbf{u}\rangle \tag{15}$$

for some random vector $\mathbf{u}$. It can be verified that

$$\nabla_{\mathbf{x}} g(\mathbf{x}) = \mathbf{J}^\top \mathbf{u} = \texttt{JacTMV}(\mathbf{x}, \mathbf{u}).$$

Hence, to test the transposed Jacobian multiplication we may apply the gradient test for $g(\mathbf{x})$ in (15) (using a random vector $\mathbf{u}$) as described in section 2.10.1.