
Selected Topics in Mathematical Optimization

K. T. Ohlhus

Jan 11, 2022

CONTENTS

I	Seminar notes	3
1	The optimization problem	5
1.1	Classes of optimizations problems	5
1.2	Local and global optima	5
1.3	Two-dimensional constrained example	8
1.4	Notation	9
2	Unconstrained minimization problems	11
2.1	Necessary conditions for a local minimum	11
2.2	Sufficient condition for a strict local minimum	13
3	Unconstrained minimization algorithms	15
3.1	Gradient methods	15
3.2	Choices of stopping criteria	24
3.3	Further approaches	25
3.4	Summary	25
3.5	Nelder-Mead method	26
4	Convex functions	37
4.1	Some special sets	37
4.2	Convex functions	44
5	Constrained minimization problems	51
5.1	The Lagrangian	51
5.2	KKT optimality conditions	51
6	Constrained minimization algorithms	55
6.1	Sequential Quadratic Programming (SQP)	55
7	Interior-Point Methods (IPM)	57
7.1	IPM: Example 1 - Barrier Method	58
7.2	IPM: Example 2 - Ill-conditioned Hessian matrix	61
7.3	A simple transformation	64
7.4	IPM Example 3 - Linear Program	66
7.5	Summarizing	70
8	Penalty methods	71
8.1	PM: Example 1	72
9	Linear Programming	75
9.1	Meeting other LP standard forms	75

II Exercises	77
10 UM01	79
11 UM02	83
12 UM03	87
13 RM01	89
13.1 Numerical experiment (only Matlab)	91
14 RM02	93
14.1 Numerical experiment (only Matlab)	95
15 RM03	97
15.1 Numerical experiment (only Matlab)	98
16 RM04	101
16.1 Sequential Quadratic Programming (SQP)	102
16.2 Numerical experiment	103
17 LP01	105
17.1 Collection of Linear Programming (LP) exercises	105
17.2 Example 1: Car factory	105
17.3 Example 2: Alloy production	107
17.4 Example 3: Oil refinery	108
17.5 Example 4: Car parts	109
17.6 Example 5: Transport cost minimization	110
18 LP02	113
18.1 Linear classification	113
19 LP03	117
19.1 Non-linear classification	117
20 MO01 - The illumination problem	123
 III Back matter	 127
21 Code Listings	129
21.1 newton_simple.m	129
21.2 nelder_mead.m	130
21.3 lp_solver.m	133
21.4 um01_experiment.m	136
21.5 um02_experiment.m	138
21.6 um03_experiment.m	140
22 Bibliography	143
Bibliography	145



- Kai Torben OHLHUS
 - カイトーベンオールフス
 - email: <mailto:ohlhus@lab.twcu.ac.jp>
 - Last update: 2022-01-11
- Based on lecture material by [C. Jansson](#) and [H. Oberle](#).

Introduction

This one semester course for students of computer science introduces “**classical**” optimization methods.

The topics covered in this course include:

- Optimality conditions for finite-dimensional (un-)constrained continuous problems
- (Un-)constrained optimization methods
- Convex analysis
- Many examples

What do I need?

- Knowledge of elementary calculus and linear algebra.
- [Matlab](#) or [GNU Octave](#).

Please note: these seminar notes are **not complete**. Several additional parts, comments, proofs, and intermediate calculations are given on the blackboard.

Modeling

- **Real world problem:** mathematical formulation, preferable as a tractable (linear or convex) problem.
- **Software:** try to classify the problem, in order to find the appropriate software.
- **Solution:** depends on your choice of the mathematical model and the input data.

Nearly all scientific and engineering fields make knowingly or unknowingly use of some kind of optimization techniques. Starting with “Operations Research”, “Machine Learning”, ... It is superfluous to write them all down.

Even nature optimizes. The basic laws of nature can be formulated in terms of **least action principles**.

Applications, examples and software

Good resource is the homepage by Hans D. Mittelman: <http://plato.asu.edu/guide.html>.

- **problems/software:** software sorted by problem to be solve
- **benchmarks:** collection of test results and performance tests
- **test cases:** example files ready to use with existing software, in different formats

- **books/tutorials:** a short list of introductory texts, some online
- **tools:** software which helps formulating an optimization problem or simplifying its solution

Literature

The *bibliography section* lists teaching books for further studies: [CT17], [LY16], [Ber16], [GNS08], and [BV04].

Part I

Seminar notes

THE OPTIMIZATION PROBLEM

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \dots, m, \\ & h_j(x) = 0, \quad j = 1, \dots, p,\end{array}$$

- $x = (x_1, x_2, \dots)$: **optimization variables**
- $f: \mathcal{X} \rightarrow \mathbb{R}$: **objective function**
- $g_i: \mathcal{X} \rightarrow \mathbb{R}, i = 1, \dots, m$: **inequality constraint functions**
- $h_j: \mathcal{X} \rightarrow \mathbb{R}, j = 1, \dots, p$: **equality constraint functions**

Decision set \mathcal{X} may be finite \mathbb{R}^n , a set of matrices, a discrete set \mathbb{Z}^n , or an infinite dimensional set.

Constraint functions $g_i(x)$ and $h_j(x)$ limit the decision set \mathcal{X} to a set of **feasible points** $x \in X \subset \mathcal{X}$.

The **Optimal solution** $x^* \in X$ has smallest value of f among all feasible points that “satisfy the constraints” :

$$\forall x \in X: f(x^*) \leq f(x).$$

1.1 Classes of optimizations problems

- Unconstrained optimization: $m = p = 0$
- Constrained optimization $m > 0$ or $p > 0$
- Linear Programming: f linear, g_i, h_j affin-linear
- Quadratic Programming: $f(x) = \frac{1}{2}x^T A x + b^T x + c$, g_i, h_j affin-linear
- Convex Optimization: f convex, g_i convex, h_j affin-linear

1.2 Local and global optima

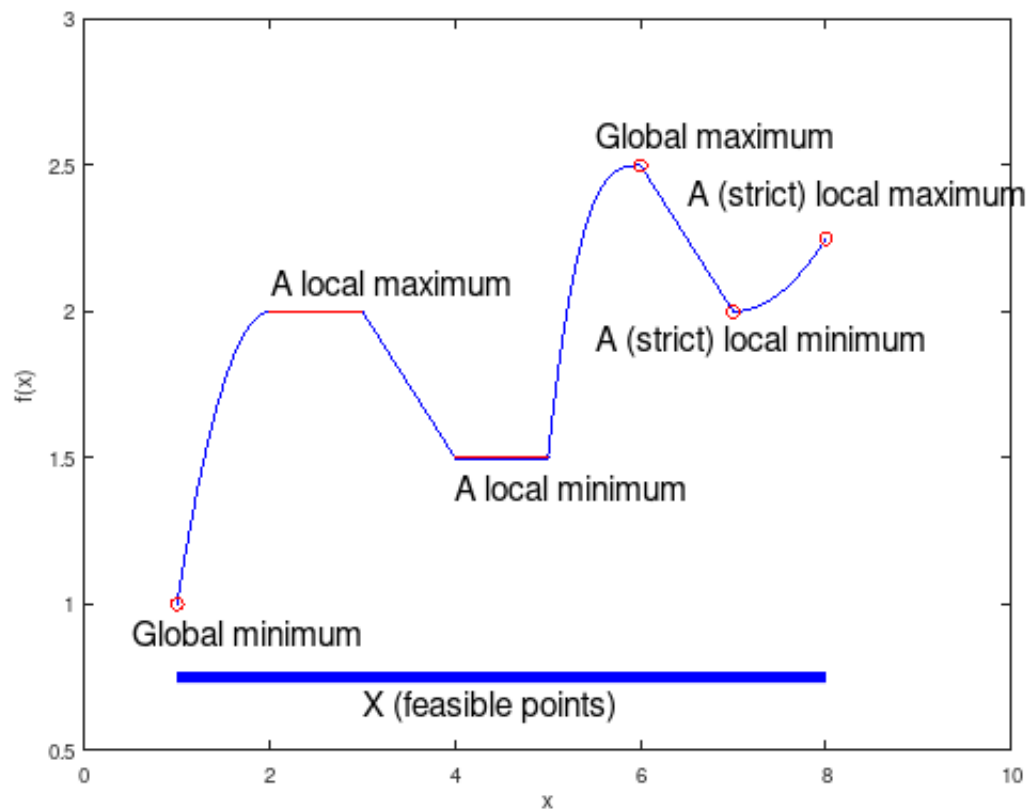
Consider the following piece-wise defined function:

```
f = { @ (x) -(x - 2) .^2 + 2, ...
      @ (x) 2, ...
      @ (x) -(x - 7) / 2, ...
      @ (x) 3/2, ...
      @ (x) (x - 6) .^3 + 5/2, ...
      @ (x) -(x - 11) / 2, ...
```

(continues on next page)

(continued from previous page)

```
@(x) (x - 7).^2 / 4 + 2;;
for i = 1:7
    fun = f{i};
    x = linspace(i, i + 1, 40);
    y = fun(x);
    plot(x, y, 'b');
    hold on;
end
xlim([0, 10]);
ylim([0.5, 3]);
xlabel('x');
ylabel('f(x)');
fmt = {'FontSize', 16};
plot(1, 1.00, 'ro'); text(0.5, 0.9, 'Global minimum', fmt{:});
plot(6, 2.50, 'ro'); text(5.5, 2.6, 'Global maximum', fmt{:});
plot(7, 2.00, 'ro'); text(5.5, 1.9, 'A (strict) local minimum', fmt{:});
plot(8, 2.25, 'ro'); text(6.5, 2.4, 'A (strict) local maximum', fmt{:});
plot(2:3, [2, 2], 'r'); text(2, 2.1, 'A local maximum', fmt{:});
plot(4:5, [3, 3] / 2, 'r'); text(4, 1.4, 'A local minimum', fmt{:});
plot([1, 8], [0.75, 0.75], 'b', 'LineWidth', 6);
text(3, 0.65, 'X (feasible points)', fmt{:});
```



Local and global optima of a two-dimensional function.

```

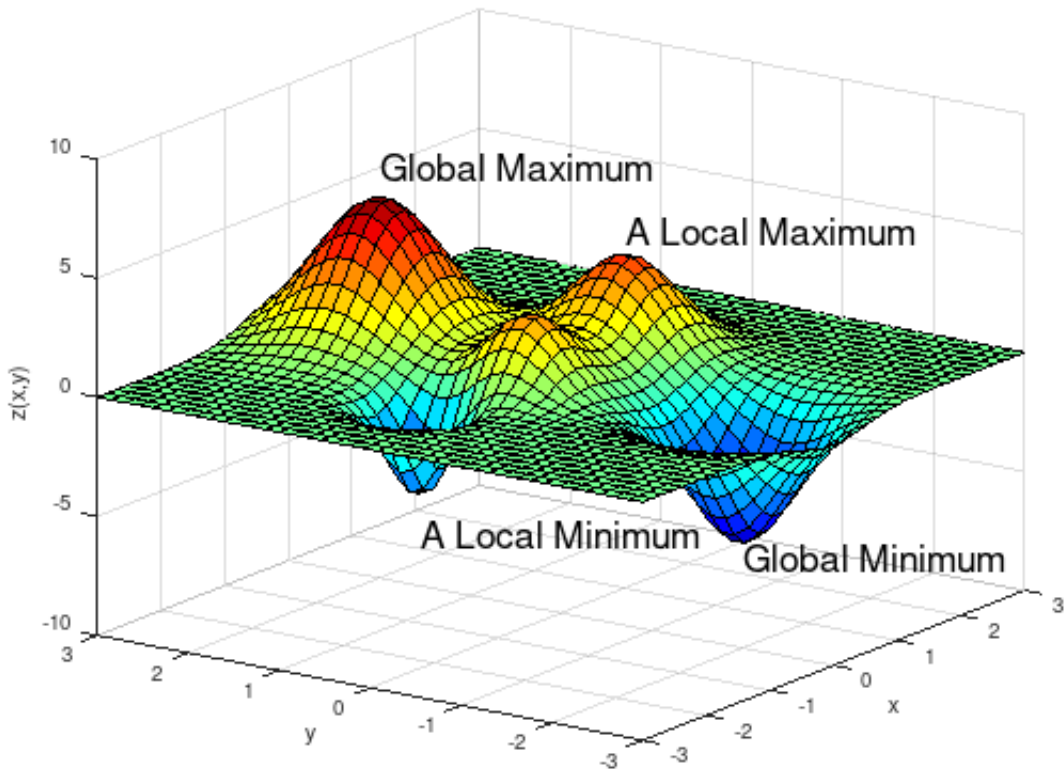
N = 3;
[X,Y] = meshgrid (linspace (-N, N, 40));

% Gaussian probability density function (PDF)
GAUSS = @(sigma, mu) 1 / (sigma * sqrt (2*pi)) * ...
    exp (-0.5 * ((X - mu(1)).^2 + (Y - mu (2)).^2) / sigma^2);

Z = 9 * GAUSS (0.6, [ 0.0,  2.0]) + 5 * GAUSS (0.5, [ 1.0,  0.0]) ...
    + 3 * GAUSS (0.4, [-0.5,  0.0]) - 3 * GAUSS (0.3, [-1.5,  0.5]) ...
    - 7 * GAUSS (0.5, [ 0.0, -2.0]);

surf (X, Y, Z);
xlabel ('x');
ylabel ('y');
zlabel ('z(x,y)');
colormap ('jet');
props = {'FontSize', 18};
text ( 0.0, -2.0, -6.2, 'Global Minimum', props{:});
text ( 0.0,  2.0,  7.2, 'Global Maximum', props{:});
text (-1.5,  0.5, -5.5, 'A Local Minimum', props{:});
text ( 1.0,  0.0,  5.0, 'A Local Maximum', props{:});
view (-55, 21)

```



1.3 Two-dimensional constrained example

A simple problem can be defined by the constraints

$$\begin{aligned}x_1 &\geq 0, \\x_2 &\geq 0, \\x_1^2 + x_2^2 &\geq 1, \\x_1^2 + x_2^2 &\leq 2,\end{aligned}$$

with an objective function to be maximized

$$f(x) = x_1 + x_2$$

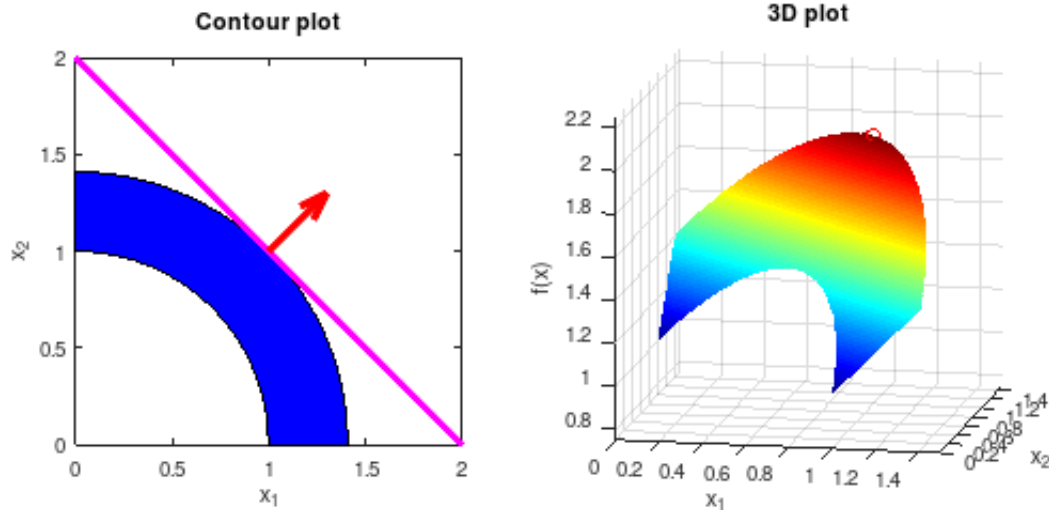
```
subplot (1, 2, 1);

% Visualize constrained set of feasible solutions (blue).
circ = @(x) sqrt (max(x)^2 - x.^2);
x_11 = 0:0.01:sqrt(2);
x_21 = circ (x_11);
x_12 = 1:-0.01:0;
x_22 = circ (x_12);
area ([x_11, x_12], [x_21, x_22], 'FaceColor', 'blue');

% Visualize level set of the objective function (magenta)
% and its scaled gradient (red arrow).
hold on;
plot ([0 2], [2 0], 'LineWidth', 4, 'm');
quiver (1, 1, 0.3, 0.3, 'LineWidth', 4, 'r');
axis equal;
xlim ([0 2]);
ylim ([0 2]);
xlabel ('x_1');
ylabel ('x_2');
title ('Contour plot');

subplot (1, 2, 2);
[X1, X2] = meshgrid (linspace (0, 1.5, 500));
FX = X1 + X2;

% Remove infeasible points.
FX((X1.^2 + X2.^2) < 1) = inf;
FX((X1.^2 + X2.^2) > 2) = inf;
surf (X1, X2, FX);
shading flat;
colormap ('jet');
hold on;
plot3 (1, 1, 2, 'ro');
axis equal;
xlabel ('x_1');
ylabel ('x_2');
zlabel ('f(x)');
title ('3D plot');
view (11, 12);
```



The intersection of the level-set of the objective function ($f(x) = 2 = \text{const.}$) and the constrained set of feasible solutions represents the solution.

$$\max f(x) = -\min -f(x)$$

In these lectures we are mainly interested in methods for computing local minima.

1.4 Notation

Vectors $x \in \mathbb{R}^n$ are column vectors. x^T is the transposed row vector.

A norm in \mathbb{R}^n is denoted by $\|\cdot\|$ and corresponds to the Euclidean norm:

$$\|x\| = \|x\|_2 = (x^T x)^{\frac{1}{2}} = \sqrt{x_1^2 + \dots + x_n^2}.$$

Is $\|\cdot\|$ a norm on \mathbb{R}^n , then

$$\|A\| := \max \{\|Ax\| : \|x\| \leq 1\}$$

defines the matrix norm (operator norm) for $A \in \mathbb{R}^{n \times n}$.

For the Euclidean vector norm this is

$$\|A\| = \sqrt{\lambda_{\max}(A^T A)} = \sigma_{\max}(A),$$

where $\lambda_{\max}(\cdot)$ denotes the maximal eigenvalue and $\sigma_{\max}(\cdot)$ the maximal singular value of a matrix.

The open ball with center $x \in \mathbb{R}^n$ and radius $\varepsilon > 0$ is denoted by

$$B_\varepsilon(x) := \{y \in \mathbb{R}^n : \|y - x\| < \varepsilon\} = \{x + \varepsilon u \in \mathbb{R}^n : \|u\| \leq 1\}.$$

The interior of a set $X \subset \mathbb{R}^n$ is denoted by $\text{int}(X)$ and the border by ∂X . For $x \in \text{int}(X)$ and $\varepsilon > 0$ there is $B_\varepsilon(x) \subset X$.

A subset $X \subset \mathbb{R}^n$ is called “**open**”, if $\text{int}(X) = X$. (“The border ∂X is not included.”)

A closed subset $X \subset \mathbb{R}^n$ is called “**compact**”, if every open cover of X has a finite sub-cover. (“No holes in X , border included.”).

In the notation $f: X \rightarrow Y$, the set X is the **domain** of f and is alternatively denoted as $\text{dom}(f)$.

Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be continuously differentiable ($f \in C^1(\mathbb{R}^n, \mathbb{R})$), then the **gradient** of f in x is denoted by the column vector:

$$\nabla f(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{pmatrix}$$

Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be twice continuously differentiable ($f \in C^2(\mathbb{R}^n, \mathbb{R})$), then the symmetric **Hessian matrix** of f in x is denoted by:

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2}{\partial x_1 \partial x_1} f(x) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f(x) & \cdots & \frac{\partial^2}{\partial x_n \partial x_n} f(x) \end{pmatrix}$$

A real symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite, if $d^T A d \geq 0$ for all $d \in \mathbb{R}^n \setminus \{0\}$ and is denoted by $A \succeq 0$.

A real symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive definite, if $d^T A d > 0$ for all $d \in \mathbb{R}^n \setminus \{0\}$ and is denoted by $A \succ 0$.

The **Landau symbol** is defined by

$$\phi(d) = o(\|d\|^k) \iff \lim_{d \rightarrow 0} \frac{\phi(d)}{\|d\|^k} = 0.$$

Taylor’s theorem (only linear and quadratic)

Let $f \in C^1(\mathbb{R}^n, \mathbb{R})$, $X \subset \mathbb{R}^n$ open, $x \in X$, $d \in \mathbb{R}^n$ and $\|d\|$ sufficiently small, then:

$$f(x + d) = f(x) + \nabla f(x)^T d + o(\|d\|).$$

Additionally, if $f \in C^2(\mathbb{R}^n, \mathbb{R})$ and for $0 < \theta < 1$, the remainder is expressible by

$$f(x + d) = f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x + \theta d) d.$$

Let $f \in C^2(\mathbb{R}^n, \mathbb{R})$, $X \subset \mathbb{R}^n$ open, $x \in X$, $d \in \mathbb{R}^n$ and $\|d\|$ sufficiently small, then:

$$f(x + d) = f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x) d + o(\|d\|^2).$$

Theorem 1: Extreme value theorem

Let $X \subset \mathbb{R}^n$ non-empty and compact and $f: \mathbb{R}^n \rightarrow \mathbb{R}$ continuous. Then there exists a **global minimum** of f over X .

If a real-valued function f is continuous on the closed interval $[a, b]$, then f must attain a maximum and a minimum, each at least once!

UNCONSTRAINED MINIMIZATION PROBLEMS

This section considers the minimization of a function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ without any constraint functions.

2.1 Necessary conditions for a local minimum

Zero slope at a local minimum x^* :

$$\nabla f(x^*) = 0$$

Theorem 2: 1st order optimality condition

Let the open set $X \subset \mathbb{R}^n$ and let $f \in C^1(\mathbb{R}^n, \mathbb{R})$. If the point $x^* \in X$ is a local minimum of f over X , then $\nabla f(x^*) = 0$.

Proof:

Let $d \in \mathbb{R}^n \setminus \{0\}$. Because X is an open set and x^* is a local minimum $x^* + td \in X$, $x^* - td \in X$, $f(x^* + td) - f(x^*) \geq 0$, and $f(x^* - td) - f(x^*) \geq 0$ hold for sufficiently small $t > 0$. Because $f \in C^1(\mathbb{R}^n, \mathbb{R})$ it follows

$$0 \leq \lim_{t \rightarrow +0} \frac{f(x^* + td) - f(x^*)}{t} = \nabla f(x^*)^T d$$

and

$$0 \leq \lim_{t \rightarrow +0} \frac{f(x^* - td) - f(x^*)}{t} = -\nabla f(x^*)^T d$$

and it follows $\nabla f(x^*) = 0$.

□

Non-negative curvature at a local minimum x^* :

$$\nabla^2 f(x^*) \succeq 0 \quad \text{is positive semi-definite}$$

Theorem 3: 2nd order optimality condition

Let the open set $X \subset \mathbb{R}^n$ and $f \in C^2(\mathbb{R}^n, \mathbb{R})$. If the point $x^* \in X$ is a local minimum of f over X , then the Hessian matrix $\nabla^2 f(x^*)$ is positive semi-definite.

Proof:

Let $d \in \mathbb{R}^n \setminus \{0\}$ and $|t|$ sufficiently small (such that $x^* + td \in X$ for all t with $|t| < t_0$).

Using Taylor's theorem (differentiation with chain-rule) and regarding $\nabla f(x^*) = 0$ gives

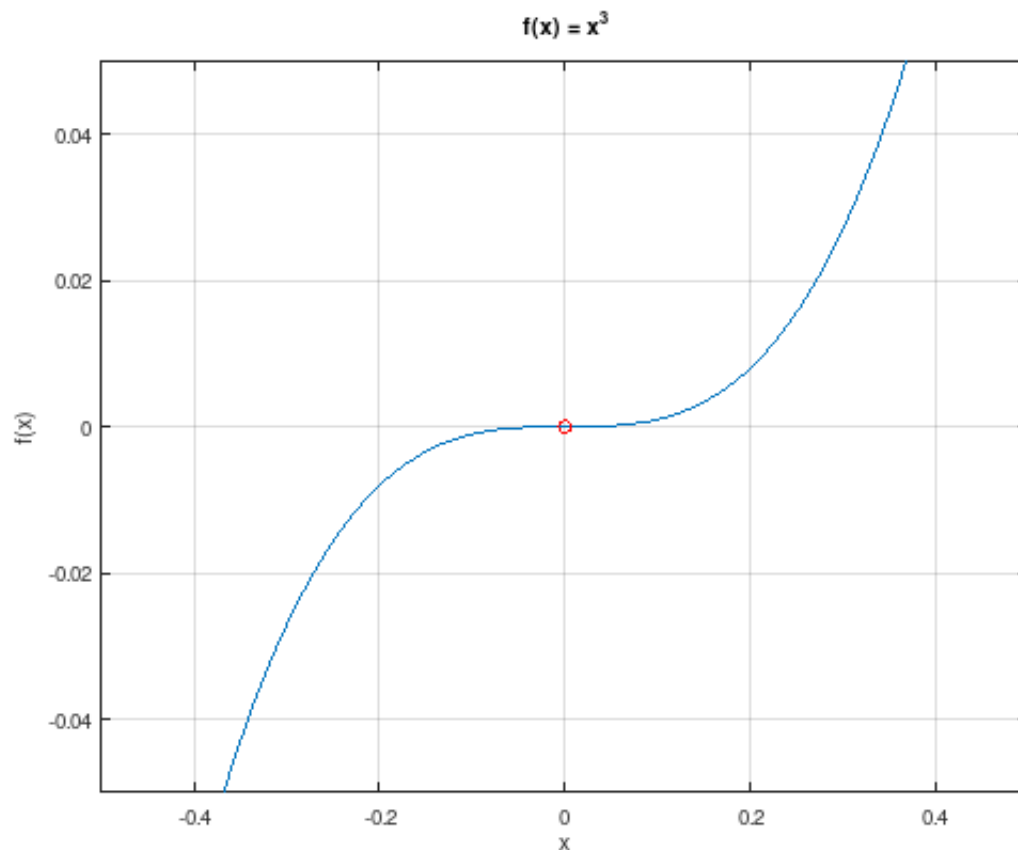
$$f(x^* + td) = f(x^*) + \frac{1}{2}t^2 d^T \nabla^2 f(x^*) d + o(t^2).$$

If $\nabla^2 f(x^*)$ was not positive semi-definite, there exists a $d \in \mathbb{R}^n \setminus \{0\}$ with $d^T \nabla^2 f(x^*) d < 0$. This is a contradiction for the minimality of x^* .

□

Note: There may exist points that satisfy the **necessary** 1st and 2nd order conditions but are not local minima. For example:

```
x = linspace (-0.5, 0.5, 50);  
plot (x, x.^3);  
hold on;  
plot (0, 0, 'ro');  
grid on;  
xlim ([-0.5, 0.5]);  
ylim ([-0.05, 0.05]);  
xlabel ('x');  
ylabel ('f(x)');  
title ('f(x) = x^3');
```



2.2 Sufficient condition for a strict local minimum

Positive curvature at a local minimum x^* (strictly convex):

$$\nabla^2 f(x^*) \succ 0 \quad \text{is positive definite}$$

Theorem 4: 2nd order sufficient optimality condition

Let $f \in C^2(\mathbb{R}^n, \mathbb{R})$ and $x^* \in X \subset \mathbb{R}^n$ with $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \succ 0$ (positive definite), then x^* is a strict local minimum of f over X .

Proof:

Application of the Rayleigh-Ritz method, for symmetric matrix $A \in \mathbb{R}^{n \times n}$ and $d \in \mathbb{R}^n \setminus \{0\}$ there is

$$\lambda_{\min}(A) \leq \frac{d^T A d}{d^T d} \leq \lambda_{\max}(A),$$

where the minimum or maximum is attained for the respective eigenvectors d of the minimal and maximal eigenvalue.

For a positive definite, symmetric matrix $A = \nabla^2 f(x^*)$ there is $\lambda := \lambda_{\min}(A) > 0$:

$$d^T \nabla^2 f(x^*) d \geq \lambda \|d\|_2^2, \quad \forall d \in \mathbb{R}^n.$$

Applying Taylor's theorem to f at x^* (with remainder) gives

$$f(x^* + d) = f(x^*) + \nabla f(x^*)^T d + \frac{1}{2} d^T \nabla^2 f(\tilde{x}) d$$

with $\tilde{x} := x^* + \theta d$, $0 < \theta < 1$.

Using $\nabla f(x^*) = 0$ and an expansion:

$$\begin{aligned} f(x^* + d) &= f(x^*) + \frac{1}{2} \underbrace{d^T \nabla^2 f(x^*) d}_{\geq \lambda \|d\|_2^2} + \frac{1}{2} \underbrace{d^T (\nabla^2 f(\tilde{x}) - \nabla^2 f(x^*)) d}_{\text{Cauchy-Schwarz inequality}} \\ &\geq f(x^*) + \frac{1}{2} (\lambda - \|\nabla^2 f(\tilde{x}) - \nabla^2 f(x^*)\|_2) \|d\|_2^2 \end{aligned}$$

Because the Hessian matrix is continuous, there is $f(x^* + d) > f(x^*)$ for all $d \neq 0$ with a sufficiently small norm.

□

Note: The sufficient condition is not necessary. See for example $f(x) = x_1^2 + x_2^4$, where $x = 0$ is a strict local minimum. However, the Hessian matrix is not positive definite.

UNCONSTRAINED MINIMIZATION ALGORITHMS

Consider the following quadratic unconstrained problem

$$\text{minimize } f(x) = \frac{1}{2}x^T A x + b^T x + c$$

where A is a symmetric $n \times n$ matrix, $x, b \in \mathbb{R}^n$, and $c \in \mathbb{R}$.

f is a C^∞ -function with

$$\nabla f(x) = Ax + b \quad \text{and} \quad \nabla^2 f(x) = A.$$

- f is convex $\iff A \succeq 0$ (positive semi-definite).
 - **Local minima are global for convex functions!** (See [Theorem 6](#)).
- f is strict convex $\iff A \succ 0$ (positive definite).
 - A invertible and $x^* = -A^{-1}b$ is the **unique global minimum**.
- $A \not\succeq 0 \implies f$ has no local minima.

3.1 Gradient methods

The general idea of gradient methods is to find the minimum of an optimization problem of the form

$$\text{minimize } f(x),$$

with $x \in \mathbb{R}^n$ and $f \in C^1(\mathbb{R}^n, \mathbb{R})$.

Many common solution strategies try to find the minimum step-wise. From a **starting point** $x^0 \in \mathbb{R}^n$ in each step $k = 0, 1, \dots$ those strategies determine

1. a **descent direction** $d^k \in \mathbb{R}^n$ and
2. a positive bounded **step size** $0 < \alpha^k < \delta$, with $\alpha^k, \delta \in \mathbb{R}$,

such that the objective function value at a new point

$$x^{k+1} = x^k + \alpha^k d^k, \quad k = 0, 1, \dots$$

is smaller than the previous one $f(x^{k+1}) < f(x^k)$.

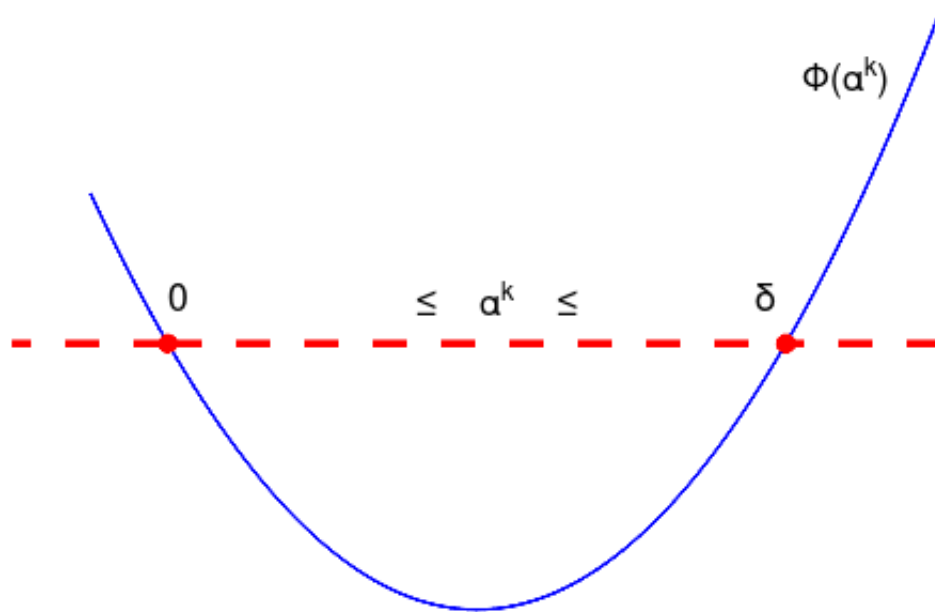
A vector $d^k \in \mathbb{R}^n$ is called **descent direction** in a point $x^k \in \mathbb{R}^n$, if there exists a $\delta > 0$ such that for all $0 < \alpha^k < \delta$ the inequality

$$f(x^k + \alpha^k d^k) < f(x^k)$$

holds.

The following figure sketches $\Phi(\alpha^k) = f(x^k + \alpha^k d^k)$, that is the objective function value as function of the step size α^k for a given descent direction d^k at a given point x^k .

```
alpha = linspace (-0.5, 5, 100);
phi = (alpha - 2).^2 + 2;
plot (alpha, phi, 'LineWidth', 2, 'b');
hold on;
plot ([-2, 0, 4, 6], [6, 6, 6, 6], 'LineWidth', 4, 'ro--');
xlim ([-1, 5]);
ylim ([ 1, 12]);
axis off;
tprops = {'FontSize', 18};
text (0.0, 6.7, '0', tprops{:});
text (1.6, 6.7, '\leq \alpha^k \leq', tprops{:});
text (3.8, 6.7, '\delta', tprops{:});
text (4.1, 10, '\Phi(\alpha^k)', tprops{:});
```



According to **Taylor's theorem**, close to a point $x^k \in \mathbb{R}^n$ a function $f \in C^1(\mathbb{R}^n, \mathbb{R})$ can be locally approximated by a linear function

$$\tilde{f}(x^k + \alpha^k d^k) := f(x^k) + \nabla f(x^k)^T (\alpha^k d^k)$$

or if $f \in C^2(\mathbb{R}^n, \mathbb{R})$ by a quadratic function

$$\tilde{f}(x^k + \alpha^k d^k) := f(x^k) + \nabla f(x^k)^T (\alpha^k d^k) + \frac{1}{2} (\alpha^k d^k)^T \nabla^2 f(x^k) (\alpha^k d^k).$$

Theorem 5: Descent direction

If $\nabla f(x^k)^T d^k < 0$ holds, then d^k is a descent direction of f in the point x^k .

Proof:

For the C^1 -function $\Phi(\alpha^k) := \tilde{f}(x^k + \alpha^k d^k)$ there is $\Phi'(0) = \nabla f(x^k)^T d^k < 0$. From this follows the assertion.

□

3.1.1 Gradient Descent

The most obvious choice for such a descent direction is $d^k := -\nabla f(x^k)$. This leads to the first algorithm for unconstrained minimization problems.

Note: Gradient descent or steepest descent performs the iteration:

$$x^{k+1} = x^k - \alpha^k \nabla f(x^k), \quad k = 0, 1, \dots$$

What can go wrong?

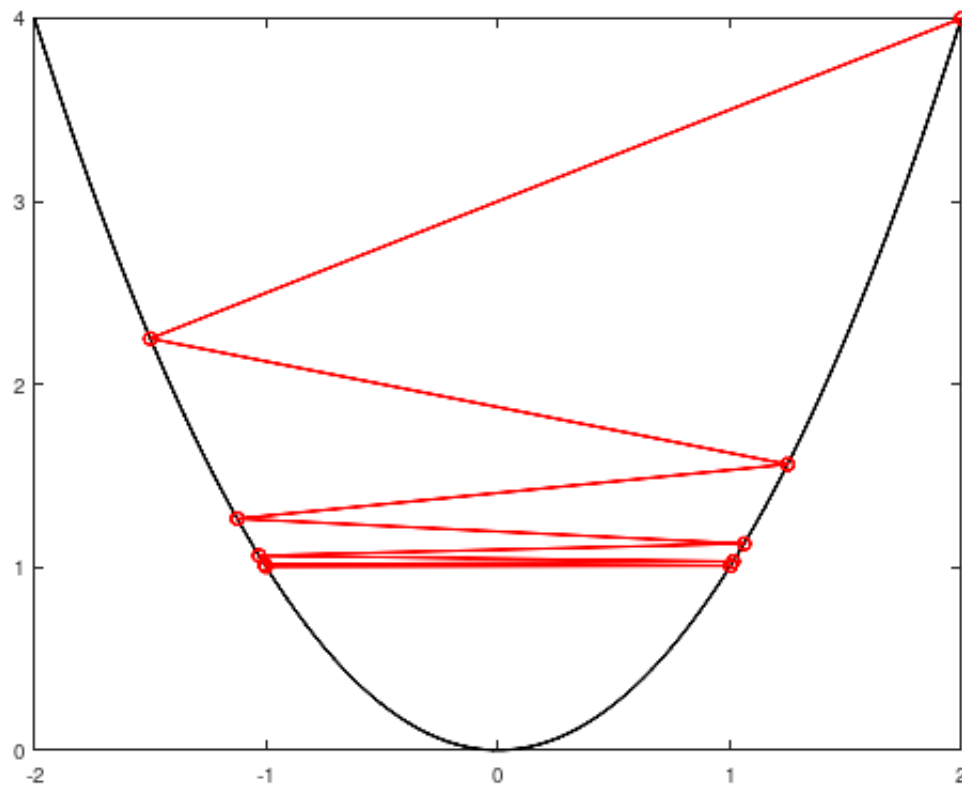
Consider the function $f(x) = x^2$, the descent directions $d^k = (-1)^{k+1}$, the step sizes $\alpha^k = 2 + \frac{3}{2^{k+1}}$ and a starting point $x_0 = 2$.

```
f = @(x) x.^2;
x = linspace(-2, 2, 100);
plot(x, f(x), 'k', 'LineWidth', 2);
hold on;

alpha = @(k) 2 + 3 / 2^(k + 1);

N = 10;
x = zeros(1, N);
x(1) = 2;
for k = 1:(N - 1)
    x(k + 1) = x(k) + alpha(k - 1) * (-1)^(k);
end

plot(x, f(x), 'ro-', 'LineWidth', 2);
```



Step size too large: convergence against pair ± 1 .

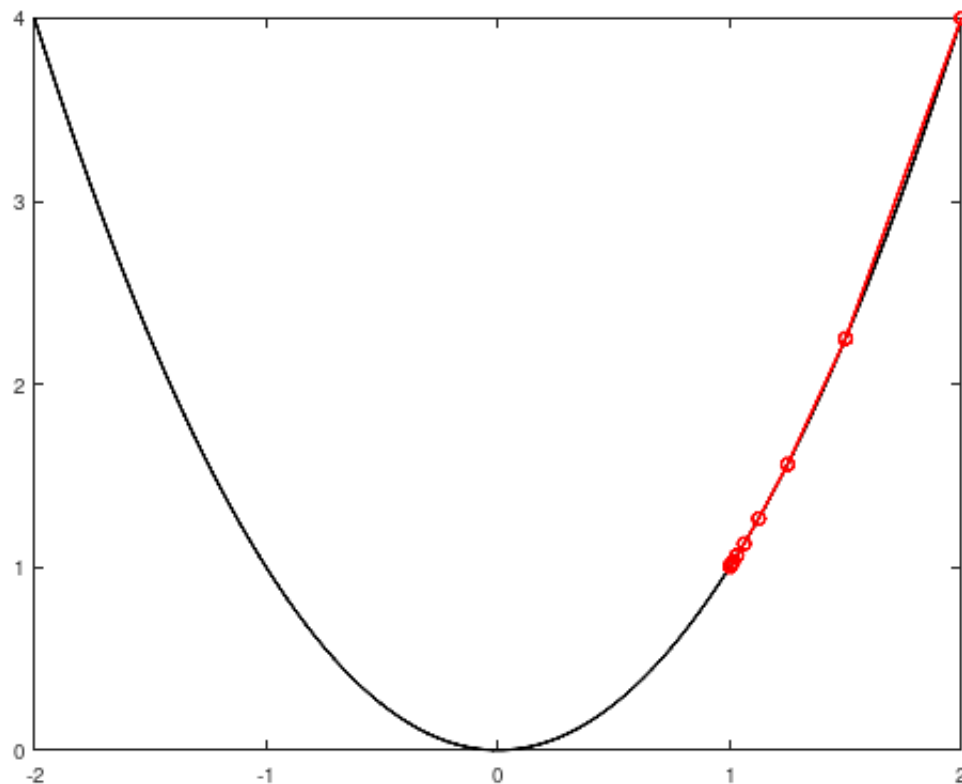
Consider the function $f(x) = x^2$, the descent directions $d^k = -1$, the step sizes $\alpha^k = \frac{1}{2^{k+1}}$ and a starting point $x_0 = 2$.

```
f = @(x) x.^2;
x = linspace (-2, 2, 100);
plot (x, f(x), 'k', 'LineWidth', 2);
hold on;

alpha = @(k) 1 / 2^(k + 1);

N = 10;
x = zeros (1, N);
x(1) = 2;
for k = 1:(N - 1)
    x(k + 1) = x(k) + alpha(k - 1) * (-1);
end

plot (x, f(x), 'ro-', 'LineWidth', 2);
```



Step size too short.

Choosing the step size

From the two (admittedly constructed) examples of the previous section it becomes apparent that not any **step size** α^k is suitable for the **gradient descent method** or other **gradient methods**.

One classical method to choose the step size is the **Armijo rule**. This method is in general not efficient, but illustrates key ideas about other step size determination methods.

A possible Matlab/Octave implementation of the Armijo rule is given in the following listing. To improve readability, the step index k has been omitted.

The inputs of the `armijo_rule` function are: a C^1 -function f , a current iteration point $x := x^k$ with its first derivative $\text{dfx} := \nabla f(x^k)$, and a descent direction $d := d^k$.

Furthermore, three parameters are defined in the function: a start value for the step size $\alpha := \alpha^k$, $\text{beta} := \beta$ with $0 < \beta < 1$, and a shrink factor $\text{tau} := \tau$ with $0 < \tau < 1$.

```
function alpha = armijo_rule (f, x, dfx, d)
    alpha = 1;      % initial value
    beta = 0.1;
    tau = 0.5;      % shrink factor
    fx = f(x);
    while (f(x + alpha * d) > fx + alpha * beta * (dfx' * d))
        alpha = tau * alpha; % shrink alpha
```

(continues on next page)

(continued from previous page)

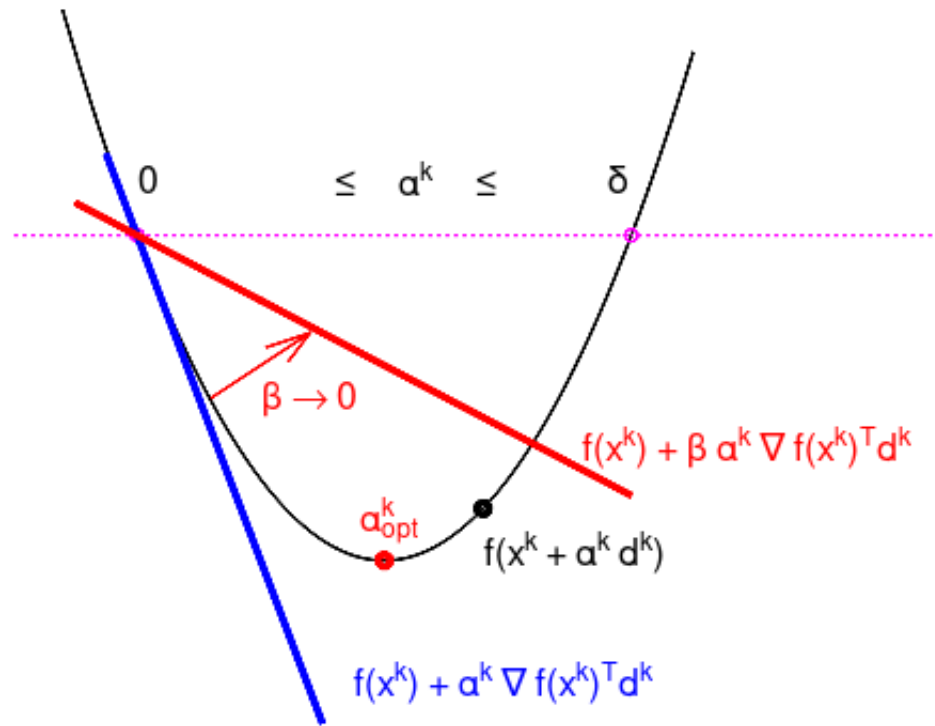
```
end
end
```

An illustration of the algorithm is given in the following Figure.

Starting with some value for α^k , the value is shrunk by τ until the objective function value $f(x^k + \alpha^k d^k)$ is below the red line, depending on β .

This ensures a decreasing objective function value. However, a bad parameter choice might slow the convergence down.

```
alpha = linspace (-0.6, 4.5, 100);
phi = (alpha - 2).^2 + 2;
plot (alpha, phi, 'LineWidth', 2, 'k');
hold on;
plot ([-2, 0, 4, 9], [6, 6, 6, 6], 'LineWidth', 2, 'mo:');
alpha = linspace (-0.25, 1.5, 10);
plot (alpha, -4 * alpha + 6, 'LineWidth', 4, 'b-');
beta = 0.2;
alpha = linspace (-0.5, 4, 10);
plot (alpha, beta * -4 * alpha + 6, 'LineWidth', 4, 'r-');
plot (2, 2, 'LineWidth', 4, 'ro');
plot (2.8, 0.8^2 + 2, 'LineWidth', 4, 'ko');
xlim ([-1, 6.5]);
ylim ([ 0, 9]);
axis off;
tprops = {'FontSize', 18};
text (0.0, 6.7, '0', tprops{:});
text (1.6, 6.7, '\leq \alpha^k \leq', tprops{:});
text (3.8, 6.7, '\delta', tprops{:});
text (2.8, 2.1, 'f(x^k) + \alpha^k d^k', tprops{:});
text (3.6, 3.4, 'f(x^k) + \beta \alpha^k \nabla f(x^k)^T d^k', ...
      tprops{:}, 'Color', 'red');
text (1.75, 0.5, 'f(x^k) + \alpha^k \nabla f(x^k)^T d^k', ...
      tprops{:}, 'Color', 'blue');
text (1.8, 2.5, '\alpha^k_{opt}', tprops{:}, 'Color', 'red');
quiver (0.6, 4, 0.8, 0.8, 'LineWidth', 2, 'r');
text (1, 4, '\beta \rightarrow 0', tprops{:}, 'Color', 'red');
```

In the optimization literature, depending on the given particular problem, there are better algorithms for choosing the step size available.

Example: gradient descent zig-zagging

Consider the quadratic unconstrained problem with the quadratic objective function

$$f(x, y) = \frac{1}{2}(x^2 + My^2) = \frac{1}{2} \begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & M \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

and with the respective gradient

$$\nabla f(x, y) = \begin{pmatrix} x \\ My \end{pmatrix}.$$

The step sizes α^k are determined by the **Armijo rule** and consider the starting point $x_0 = (M, 1)^T$ for different values of M .

```
MM = [1, 5];
for i = 1:2
    subplot(2, 1, i);
    M = MM(i);
    f = @(x,y) (x.^2 + M * y.^2) / 2;
    [x, y] = meshgrid(linspace(-6, 6, 40), ...
                      linspace(-3, 3, 40));
```

(continues on next page)

(continued from previous page)

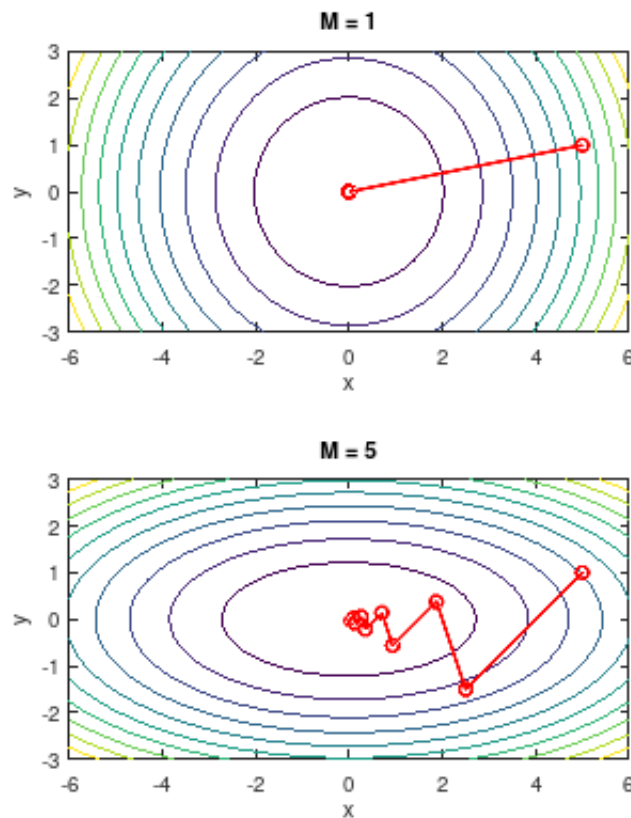
```

contour (x, y, f(x,y));
hold on;

f = @(x) (x(1,:).^2 + M * x(2,:).^2) / 2;
df = @(x) [x(1); M * x(2)]; % Gradient

N = 10;
x = zeros (2, N);
x(:,1) = [5; 1];
for k = 1:(N - 1)
    d = df(x(:,k));
    alpha = armijo_rule (f, x(:,k), d, -d);
    x(:,k + 1) = x(:,k) - alpha * d;
end
plot3 (x(1,:), x(2,:), f(x), 'ro-', 'LineWidth', 2);
axis equal;
title (['M = ', num2str(M)]);
xlabel ('x');
ylabel ('y');
end

```



- Fast convergence, if M close to 1. The gradient “points” in the direction of the global minimum.
- Slow zig-zagging, if $M \gg 1$ or $M \ll 1$. In each step the gradient “points away” from the global minimum.
- **Note:** with optimal step size, the convergence behavior is similar.

3.1.2 Newton's method

A more sophisticated choice for the descent direction d^k can be obtained from the quadratic Taylor approximation.

Like in the example from the introduction this is a quadratic unconstrained problem and it has a unique global minimum, if the Hessian matrix is positive definite.

Thus deriving the quadratic Taylor approximation to d^k and choosing without loss of generality $\alpha^k = 1$, one obtains for the first order necessary condition

$$\nabla \tilde{f}(x^k + d^k) := \nabla f(x^k) + \nabla^2 f(x^k) d^k = 0.$$

Finally, the descent direction d^k is the solution of the following linear system of equations:

$$\nabla^2 f(x^k) d^k = -\nabla f(x^k).$$

Note: Newton's method performs the iteration:

$$x^{k+1} = x^k - \alpha^k (\nabla^2 f(x^k))^{-1} \nabla f(x^k), \quad k = 0, 1, \dots$$

Example: Newton's method

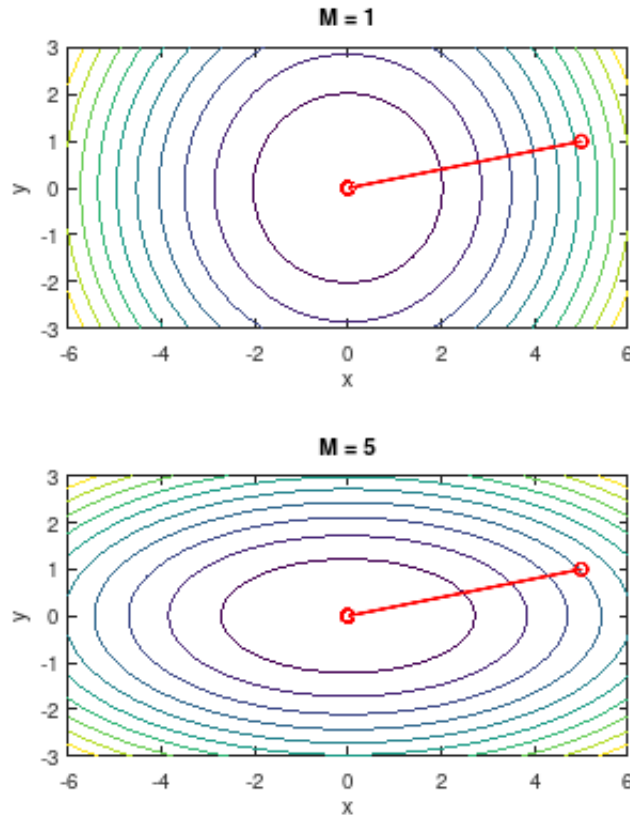
Consider again the previous quadratic unconstrained problem with the objective function $f(x, y) = \frac{1}{2}(x^2 + My^2)$.

The step sizes are $\alpha^k = 1$ and consider the starting point $x_0 = (M, 1)^T$ for different values of M .

```
MM = [1, 5];
for i = 1:2
    subplot(2, 1, i);
    M = MM(i);
    f = @(x,y) (x.^2 + M * y.^2) / 2;
    [x, y] = meshgrid(linspace(-6, 6, 40), ...
                      linspace(-3, 3, 40));
    contour(x, y, f(x,y));
    hold on;

    f = @(x) (x(1,:).^2 + M * x(2,:).^2) / 2;
    fgrad = @(x) [x(1); M * x(2)]; % Gradient
    fhess = [1, 0; 0, M]; % Hessian matrix

    N = 10;
    x = zeros(2, N);
    x(:,1) = [5; 1];
    for k = 1:(N - 1)
        d = fhess \ -fgrad(x(:,k));
        x(:,k + 1) = x(:,k) + d;
    end
    plot3(x(1,:), x(2,:), f(x), 'ro-', 'LineWidth', 2);
    axis equal;
    title(['M = ', num2str(M)]);
    xlabel('x');
    ylabel('y');
end
```



While the gradient descent method converged in the latter case zig-zagging, Newton's method seems unaffected by the elliptic shape of the objective function and converges in one step.

This beneficial convergence behavior comes at the price of having to evaluate the Hessian matrix and solving a linear system of equations in each iteration step.

3.2 Choices of stopping criteria

As important as good convergence towards the minimal point, is to decide when to stop the optimization algorithm.

In literature there is a huge choice of stopping criteria, just to list a few:

1. $f(x^{k-1}) - f(x^k) \leq \text{TOL}(1 + |f(x^k)|)$
2. $\|x^{k-1} - x^k\| \leq \text{TOL}(1 + \|x^k\|)$
3. $\|\nabla f(x^k)\| \leq \text{TOL}(1 + |f(x^k)|)$
4. $\|\nabla f(x^k)\| \leq \text{machine precision}$
5. $k \geq k_{\max}$

Criteria 1 to 3 monitor the progress of the algorithm, while criteria 4 and 5 can be regarded as “emergency breaks”. It is useful to implement criteria like 4 and 5 to avoid **infinite loops** or to stop stuck algorithm runs.

However, it often depends on the given problem which other stopping criteria are useful. For example, if the objective function is expensive to evaluate, “cheap” stopping criteria which do not cause computational overhead or are computed by the algorithms anyways might be preferable.

3.3 Further approaches

Other gradient methods follow in principal the form

$$x^{k+1} = x^k - \alpha^k D^k \nabla f(x^k), \quad k = 0, 1, \dots,$$

where $D^k \succ 0$ is a symmetric positive definite matrix.

For example:

- **Diagonally scaled steepest descent**

D^k is a diagonal approximation to $(\nabla^2 f(x^k))^{-1}$

- **Modified Newton' s method**

$$D^k = (\nabla^2 f(x^0))^{-1}, \quad k = 0, 1, \dots$$

- **Discretized Newton' s method**

$$D^k = (H(x^k))^{-1}, \quad k = 0, 1, \dots,$$

where $H(x^k)$ is a finite difference based approximation of $\nabla^2 f(x^k)$.

For example the **forward-** and **central finite difference** formulas:

$$\frac{\partial f(x^k)}{\partial x_i} \approx \frac{1}{h} (f(x^k + h e_i) - f(x^k)),$$

$$\frac{\partial f(x^k)}{\partial x_i} \approx \frac{1}{2h} (f(x^k + h e_i) - f(x^k - h e_i)),$$

where e_i is the i -th unit vector and h an appropriate small quantity taller than the machine precision.

- **Quasi Newton Methods**

Another strategy to avoid the computationally expensive evaluation of the whole Hessian matrix in each step is to **update the existing Hessian matrix** in the so-called **Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm**:

Determine a new step x^{k+1} from solving the Newton direction with Hessian matrix approximation D^k . Then set $q^k = x^{k+1} - x^k$ and $p^k = \nabla f(x^{k+1}) - \nabla f(x^k)$ and finally update the current Hessian matrix approximation

$$D^{k+1} = D^k + \frac{p^k p^{kT}}{p^{kT} q^k} - \frac{D^k q^k q^{kT} D^k}{q^{kT} D^k q^k}.$$

3.4 Summary

The zoo of optimization algorithms and their flavors grows steadily. It is beyond the scope of a seminar to cover them all in detail.

Gradient descent and **Newton' s methods** are well-studied in the optimization literature and have been presented without detailed analysis or proof of their convergence behavior.

Summarizing, typical pitfalls for **gradient methods** are:

- Step size too long or short (“damping” , Armijo rule, etc.).
- Convergence to stationary points ($\nabla f(x) = 0$) that are no local minima (e.g. saddle points).
- Stagnation when starting at a stationary point.

- Alternating sequence of iteration points.
- No or slow convergence if started “too far” away from the local optimum.
- In case of Newton’s method: $\nabla^2 f(x^k) \not\prec 0$, convergence unknown.

3.5 Nelder-Mead method

In the previous subsection, the **gradient methods** require the objective function to be once or twice continuously differentiable.

The **Nelder-Mead method** [NM65] described in this subsection, requires the objective function to be **continuous only**. Therefore it is an example of a **derivative-free** optimization method.

This method is implemented in the Matlab/Octave function `fminsearch`. See [LRWW98] and [PCB02] for modifications and references.

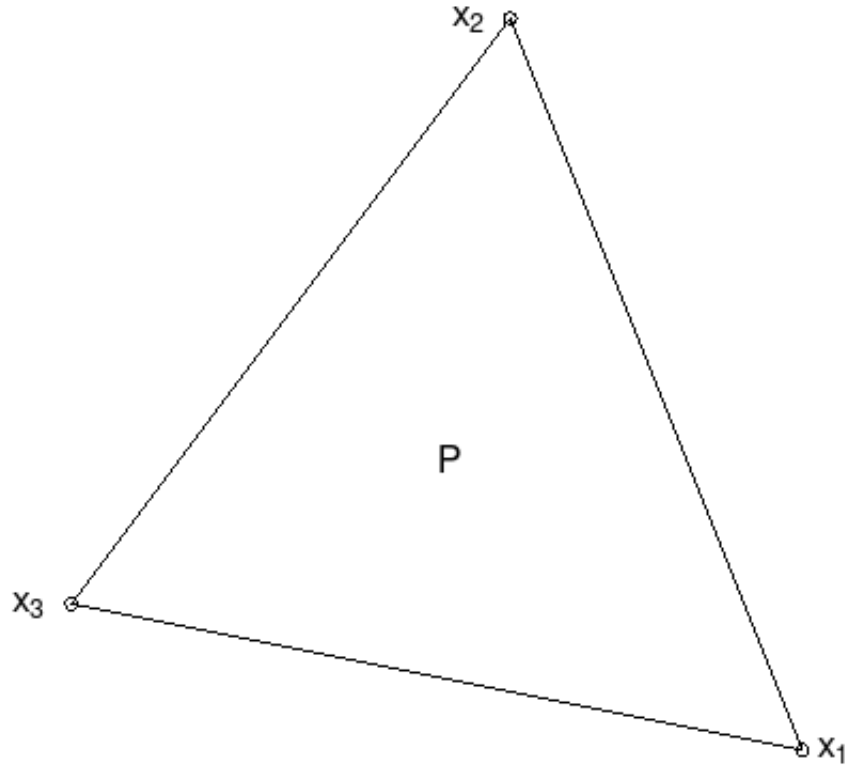
Basic concept of this method is a **simplex (polyhedron)**, here defined as a set of $n + 1$ linear independent points $P = \{x_1, x_2, \dots, x_{n+1}\} \subset \mathbb{R}^n$ (the vertices of the simplex).

In geometry the convex hull of P is usually defined as simplex. For dimension $n = 2$, the simplex is a **non-degenerate triangle**, as in the following figure. For dimension $n = 3$, a non-degenerate tetrahedron.

```
x1 = [ 0.0, 0.0];
x2 = [-0.4, 1.0];
x3 = [-1.0, 0.2];
x = [x1; x2; x3; x1];
x_label = {'x_1', 'x_2', 'x_3'};
plot (x(:,1), x(:,2), 'ko-');
hold on;

tprops = {'FontSize', 18};
for i = 1:3
    if (x(i,1) < 0)
        x_offset = -0.08;
    else
        x_offset = 0.02;
    end
    text (x(i,1) + x_offset, x(i,2), x_label{i}, tprops{:});
end

text (-0.5, 0.4, 'P', tprops{:});
axis equal;
axis off;
```



3.5.1 Iteration step of the simplex method ($n \geq 2$)

1. Compute the following indices $s, a, b \in \{1, \dots, n+1\}$:
 - $x^s := \operatorname{argmax}\{f(x) : x \in P\}$ (worst point)
 - $x^a := \operatorname{argmax}\{f(x) : x \in P, a \neq s\}$ (second worst point)
 - $x^b := \operatorname{argmin}\{f(x) : x \in P, b \neq s, b \neq a\}$ (best point)
1. Compute the **centroid** m of the n best points

$$m := \frac{1}{n} \sum_{\substack{i=0 \\ i \neq s}}^{n+1} x^i,$$

and the point r **reflected** at the center m

$$r := m + \alpha(m - x^s),$$

with $\alpha = 1$.

```
function plot_line (points, LineSpec)
    plot (points(:,1), points(:,2), LineSpec{:});
end
```

(continues on next page)

(continued from previous page)

```

function plot_triangle (points, LineSpec)
    % Repeat first point.
    plot_line ([points; points(1,:)], LineSpec);
end

xb = [ 0.0, 0.0];
xa = [-0.4, 1.0];
xs = [-1.0, 0.2];
m = xb + 0.5 * (xa - xb);
r = m +      (m - xs);

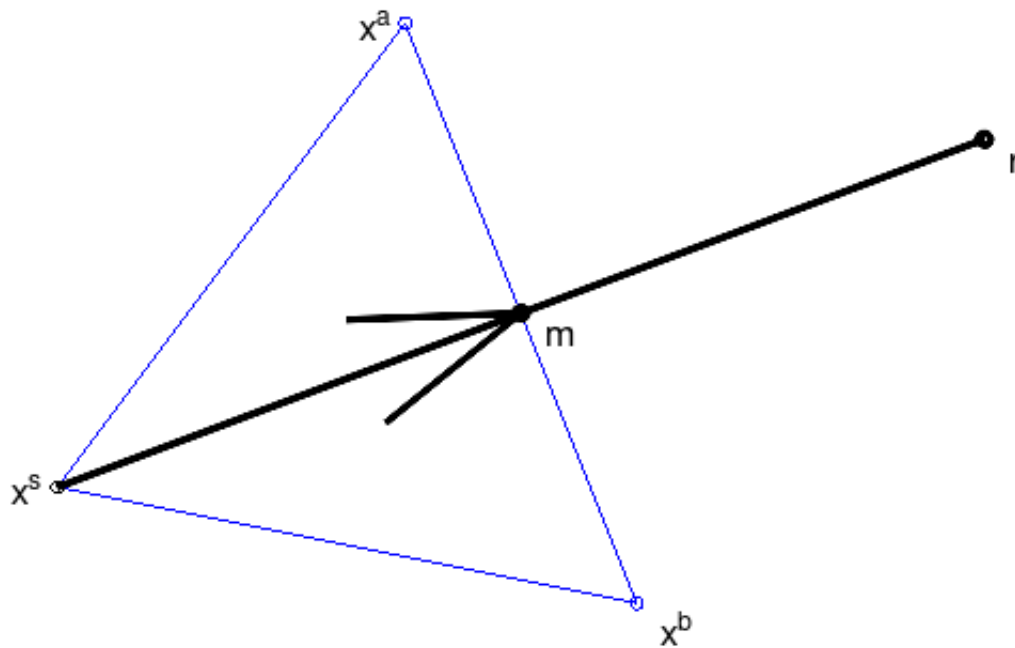
points = [xb; xa; xs; m; r];
plabel = {'x^b', 'x^a', 'x^s', 'm', 'r'};

plot_triangle ([xb; xa; xs], {'bo-'});
hold on;
quiver (xs(1), xs(2), m(1) - xs(1), m(2) - xs(2), 'ko-', 'LineWidth', 4);
plot_line ([r ; m], {'ko-', 'LineWidth', 4});

tprops = {'FontSize', 18};
for i = 1:length (plabel)
    if (points(i,1) < -0.3)
        x_offset = -0.08;
        y_offset = 0.0;
    else
        x_offset = 0.04;
        y_offset = -0.04;
    end
    text (points(i,1) + x_offset, points(i,2) + y_offset, plabel{i}, tprops{:});
end

axis equal;
axis off;

```

Case 1: Reflect

“New point is good.”

Replace x^s by r , if $f(x^b) \leq f(r) \leq f(x^a)$ and **case 2** does not apply.

```
function plot_line (points, LineSpec)
    plot (points(:,1), points(:,2), LineSpec{:});
end

function plot_triangle (points, LineSpec)
    % Repeat first point.
    plot_line ([points; points(1,:)], LineSpec);
end

xb = [ 0.0, 0.0];
xa = [-0.4, 1.0];
xs = [-1.0, 0.2];
m = xb + 0.5 * (xa - xb);
r = m + (m - xs);

points = [xb; xa; xs; m; r];
plabel = {'x^b', 'x^a', 'x^s', 'm', 'r'};

plot_triangle ([xb; xa; xs], {'ko--'});
```

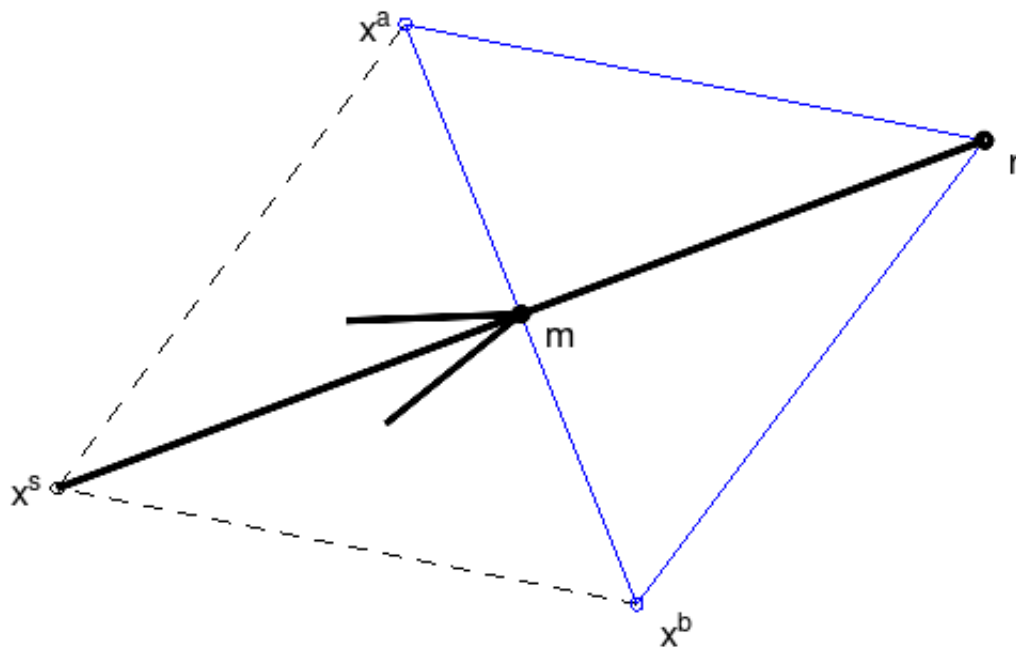
(continues on next page)

(continued from previous page)

```
hold on;
plot_triangle ([xb; xa; r] , {'bo-'});
quiver (xs(1), xs(2), m(1) - xs(1), m(2) - xs(2), 'ko-', 'LineWidth', 4);
plot_line ([r ; m], {'ko-', 'LineWidth', 4});

tprops = {'FontSize', 18};
for i = 1:length(plabel)
    if (points(i,1) < -0.3)
        x_offset = -0.08;
        y_offset = 0.0;
    else
        x_offset = 0.04;
        y_offset = -0.04;
    end
    text (points(i,1) + x_offset, points(i,2) + y_offset, plabel{i}, tprops{:});
end

axis equal;
axis off;
```



Case 2: Expand

“New point is excellent.”

If $f(r) < f(x^b)$, compute **expanded point**

$$e := m + \beta(m - x^s),$$

with $\beta > \alpha$, here $\beta = 2$.

Replace x^s by e , if $f(e) < f(r)$, otherwise replace x^s by r .

```
function plot_line (points, LineSpec)
    plot (points(:,1), points(:,2), LineSpec{:});
end

function plot_triangle (points, LineSpec)
    % Repeat first point.
    plot_line ([points; points(1,:)], LineSpec);
end

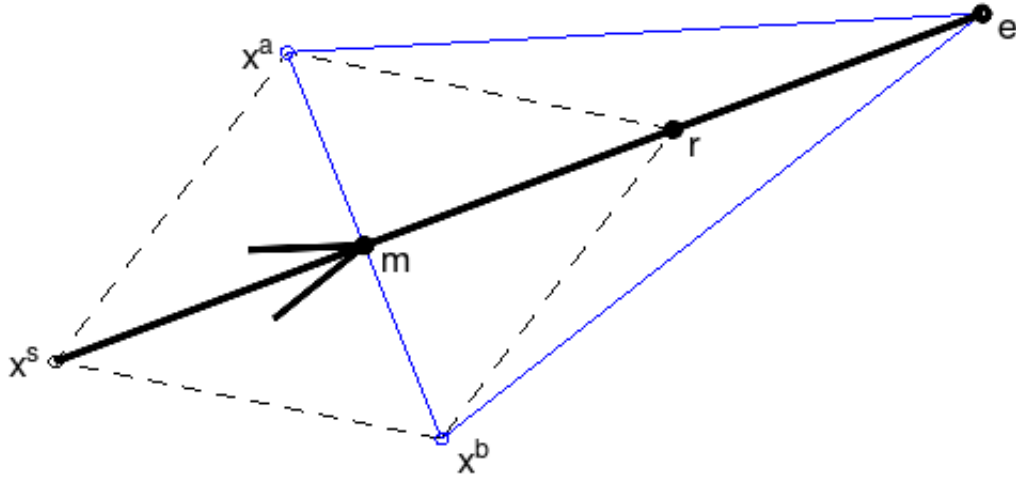
xb = [ 0.0, 0.0];
xa = [-0.4, 1.0];
xs = [-1.0, 0.2];
m = xb + 0.5 * (xa - xb);
r = m + 1.0 * (m - xs);
e = m + 2.0 * (m - xs);

points = [xb; xa; xs; m; r; e];
plabel = {'x^b', 'x^a', 'x^s', 'm', 'r', 'e'};

plot_triangle ([xb; xa; xs], {'ko--'});
hold on;
plot_triangle ([xb; xa; r] , {'ko--'});
plot_triangle ([xb; xa; e] , {'bo-'});
quiver (xs(1), xs(2), m(1) - xs(1), m(2) - xs(2), 'ko-', 'LineWidth', 4);
plot_line ([e; r; m], {'ko-', 'LineWidth', 4});

tprops = {'FontSize', 18};
for i = 1:length(plabel)
    if (points(i,1) < -0.3)
        x_offset = -0.12;
        y_offset = 0.0;
    else
        x_offset = 0.04;
        y_offset = -0.04;
    end
    text (points(i,1) + x_offset, points(i,2) + y_offset, plabel{i}, tprops{:});
end

axis equal;
axis off;
```



```
function plot_line (points, LineSpec)
    plot (points(:,1), points(:,2), LineSpec{:});
end
```

(continues on next page)

(continued from previous page)

```

function plot_triangle (points, LineSpec)
    % Repeat first point.
    plot_line ([points; points(1,:)], LineSpec);
end

xb = [ 0.0, 0.0];
xa = [-0.4, 1.0];
xs = [-1.0, 0.2];
m = xb + 0.5 * (xa - xb);
r = m + 1.0 * (m - xs);
co = m + 0.5 * (m - xs);
ci = m - 0.5 * (m - xs);

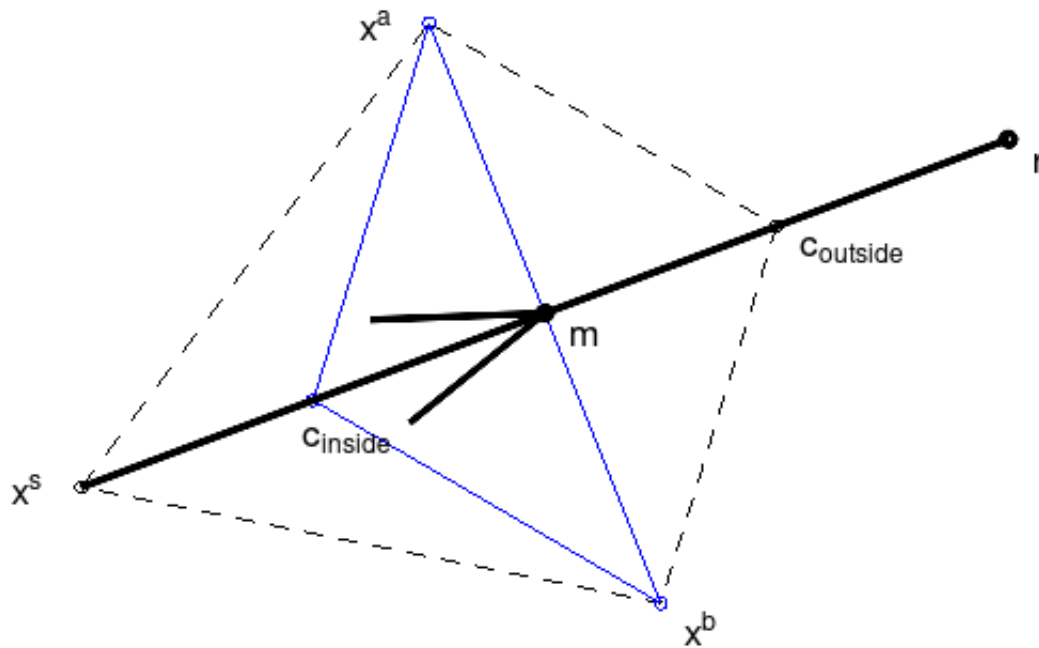
points = [xb; xa; xs; m; r; co; ci];
plabel = {'x^b', 'x^a', 'x^s', 'm', 'r', 'c_{outside}', 'c_{inside}'};

plot_triangle ([xb; xa; xs], {'ko--'});
hold on;
plot_triangle ([xb; xa; co] , {'ko--'});
plot_triangle ([xb; xa; ci] , {'bo--'});
quiver (xs(1), xs(2), m(1) - xs(1), m(2) - xs(2), 'ko-', 'LineWidth', 4);
plot_line ([r; m], {'ko-', 'LineWidth', 4});

tprops = {'FontSize', 18};
for i = 1:length (plabel)
    if (points(i,1) < -0.3)
        x_offset = -0.12;
        y_offset = 0.0;
    else
        x_offset = 0.04;
        y_offset = -0.04;
    end
    if (i == 7) % c_{inner}
        x_offset = -0.02;
        y_offset = -0.07;
    end
    text (points(i,1) + x_offset, points(i,2) + y_offset, plabel{i}, tprops{:});
end

axis equal;
axis off;

```



This concludes one step of the simplex method. There is no guarantee for convergence to a local minimum.

Potential stopping criteria are:

-

$$\forall i, j \in \{1, \dots, n+1\}: \quad \|x^i - x^j\| \leq \text{TOL}.$$

-

$$\frac{1}{n+1} \sum_{i=1}^{n+1} (f(x^i) - f(m))^2 \leq (\text{TOL})^2.$$

-

$$\frac{1}{n+1} \sum_{i=1}^{n+1} (f(x^i) - \bar{f})^2 \leq (\text{TOL})^2, \quad \bar{f} := \frac{1}{n+1} \sum_{i=1}^{n+1} f(x^i).$$

3.5.2 Start simplex

Given one vertex x^1 and one edge length l , a regular simplex can be constructed as follows:

-

$$p := \frac{l}{\sqrt{2}} \frac{\sqrt{n+1} - 1}{n}$$

-

$$q := p(1, \dots, 1)^T \in \mathbb{R}^n$$

-

$$x^j := x^1 + q + \frac{l}{\sqrt{2}} e^{j-1}, \quad j = 2, \dots, n+1,$$

where e^1, \dots, e^n are the standard unit vectors in \mathbb{R}^n .

CONVEX FUNCTIONS

This section is devoted to **convex sets** and **convex functions**, which have useful properties for optimization algorithms.

4.1 Some special sets

4.1.1 Lines and line segments

Points y on a **line** through two points $x_1 \neq x_2$ in \mathbb{R}^n can be expressed by

$$y = \theta x_1 + (1 - \theta)x_2,$$

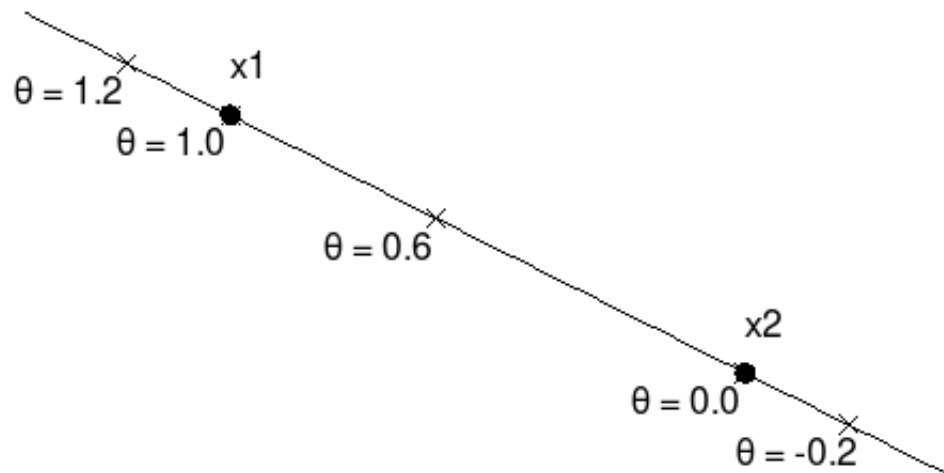
or

$$y = x_2 + \theta(x_1 - x_2),$$

where $\theta \in \mathbb{R}$.

For a **line segment**, θ is limited to $0 \leq \theta \leq 1$.

```
x1 = [0.0; 0.5];
x2 = [1.0; 0.0];
theta = linspace (-0.4, 1.4, 100);
y = @(t) x2 + t .* (x1 - x2);
yy = y(theta);
plot (yy(1,:), yy(2,:), 'k');
hold on;
tprops = {'FontSize', 18};
mprops = {'MarkerSize', 9, ...
          'MarkerFaceColor', 'black'};
for t = [1.2, 1, 0.6, 0, -0.2]
    yy = y(t);
    plot (yy(1), yy(2), 'kx', mprops{:});
    text (yy(1) - 0.22, yy(2) - 0.05, ...
          sprintf ('\theta = %.1f', t), tprops{:});
end
plot (x1(1), x1(2), 'ko', mprops{:});
text (x1(1), x1(2) + 0.1, 'x1', tprops{:});
plot (x2(1), x2(2), 'ko', mprops{:});
text (x2(1), x2(2) + 0.1, 'x2', tprops{:});
axis equal;
axis off;
```



4.1.2 Affine sets

An **affine set** contains the **line** through any two distinct points in the set.

For example the solution set of linear equations $\{x | Ax = b\}$ is affine.

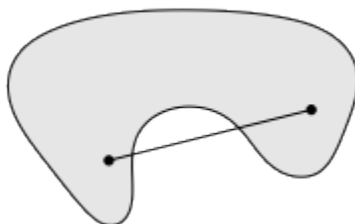
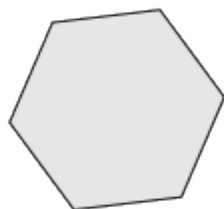
Conversely, every affine set can be expressed as solution set of a system of linear equations.

4.1.3 Convex sets

A **convex set** C contains the **line segment** through any two distinct points in the set.

$$x_1, x_2 \in C, \quad 0 \leq \theta \leq 1 \quad \implies \quad \theta x_1 + (1 - \theta)x_2 \in C.$$

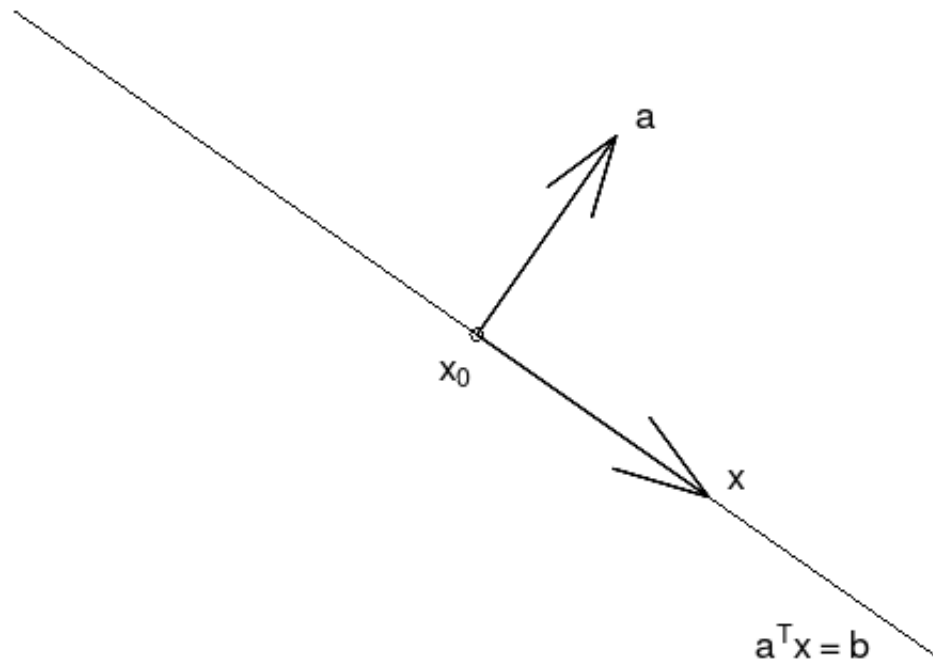
Examples (from [BV04] (p. 24): first set is convex, the latter two not)



4.1.4 Hyperplanes

Affine and convex sets of the form $\{x \in \mathbb{R}^n : a^T x = b\}$ with normal vector $a \neq 0$.

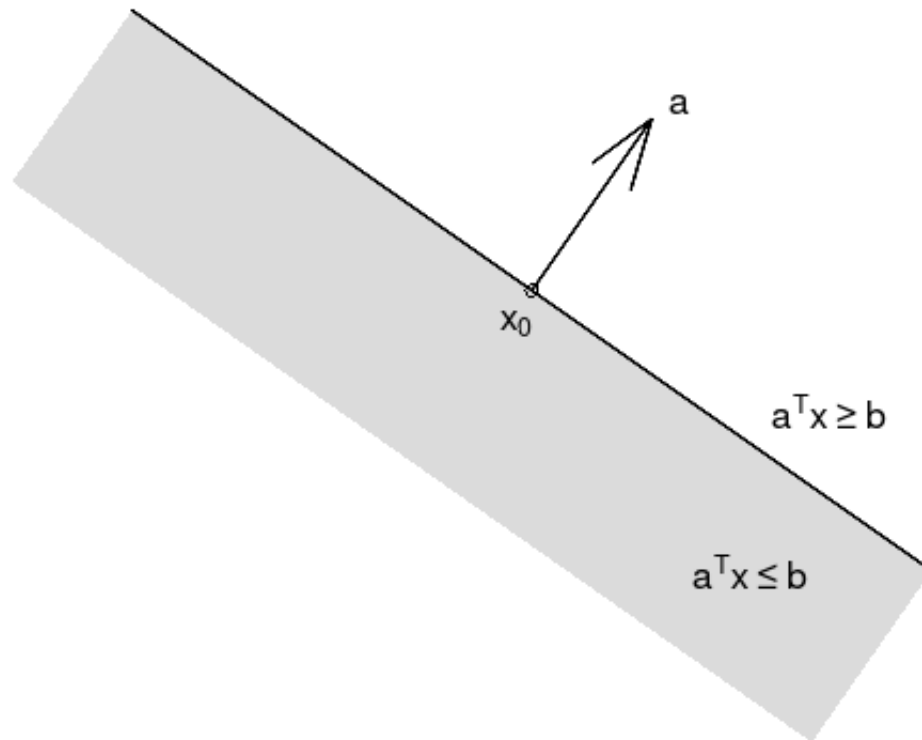
```
x1 = [0, 0.7];
x2 = [1, 0.0];
x0 = x1 + 0.5 .* (x2 - x1);
x = 0.25 .* (x2 - x1);
a = [0.15, 0.15 / x1(2)];
plot ([x1(1), x2(1)], [x1(2), x2(2)], 'k-');
hold on;
plot (x0(1), x0(2), 'ko');
quiver (x0(1), x0(2), a(1), a(2), 'k', 'LineWidth', 2);
quiver (x0(1), x0(2), x(1), x(2), 'k', 'LineWidth', 2);
tprops = {'FontSize', 18};
text (x0(1) - 0.04, x0(2) - 0.04, 'x_0', tprops{:});
text (x0(1) + a(1) + 0.02, x0(2) + a(2) + 0.02, 'a', tprops{:});
text (x0(1) + x(1) + 0.02, x0(2) + x(2) + 0.02, 'x', tprops{:});
text (x2(1) - 0.2, x2(2) + 0.02, 'a^T x = b', tprops{:});
axis equal;
axis off;
```



4.1.5 Halfspaces

Convex sets of the form $\{x \in \mathbb{R}^n : a^T x \leq b\}$ with normal vector $a \neq 0$.

```
x1 = [0, 0.7];
x2 = [1, 0.0];
x0 = x1 + 0.5 .* (x2 - x1);
a = [0.15, 0.15 / x1(2)];
x3 = x1 - a;
x4 = x2 - a;
fill ([x1(1), x2(1), x4(1), x3(1)], [x1(2), x2(2), x4(2), x3(2)], ...
      [220, 220, 220] / 256, 'EdgeColor', 'none');
hold on;
plot ([x1(1), x2(1)], [x1(2), x2(2)], 'k-', 'LineWidth', 2);
plot (x0(1), x0(2), 'ko');
quiver (x0(1), x0(2), a(1), a(2), 'k', 'LineWidth', 2);
tprops = {'FontSize', 18};
text (x0(1) - 0.04, x0(2) - 0.04, 'x_0', tprops{:});
text (x0(1) + a(1) + 0.02, x0(2) + a(2) + 0.02, 'a', tprops{:});
text (x2(1) - 0.2, x2(2) + 0.2, 'a^T x \geq b', tprops{:});
text (x2(1) - 0.3, x2(2), 'a^T x \leq b', tprops{:});
axis equal;
axis off;
```



4.1.6 Ellipsoid

Convex sets of the form

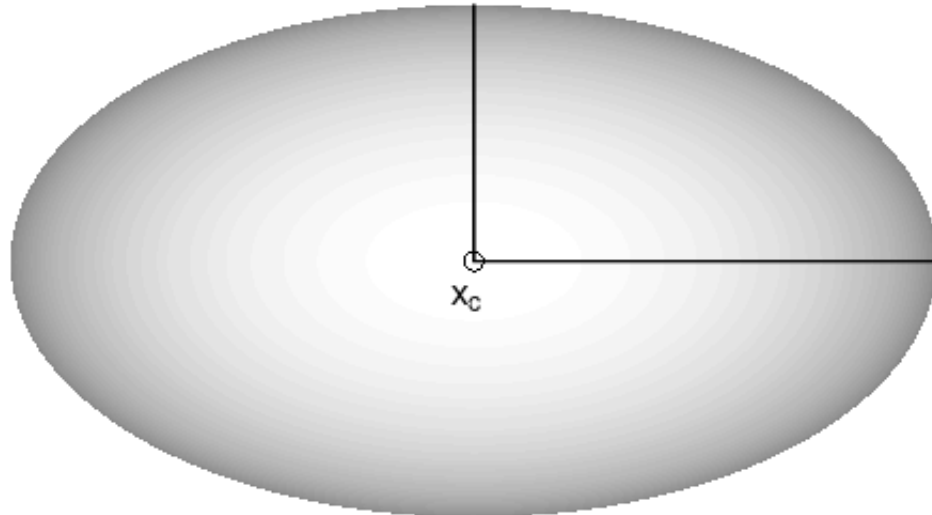
$$\{x \in \mathbb{R}^n : (x - x_c)^T P^{-1} (x - x_c) \leq 1\},$$

with center $x_c \in \mathbb{R}^n$ and $P \succ 0$ (symmetric positive definite) or

$$\{x_c + Au \in \mathbb{R}^n : \|u\| \leq 1\},$$

with A square and non-singular.

```
rx = 2;
ry = 0.7;
rz = 1;
[x, y, z] = ellipsoid (0, 0, 0, rx, ry, rz, 200);
colormap ('gray');
surf (x, y, z);
hold on;
plot3 ([0, rx], [0, 0], [1, 1], 'k-', 'LineWidth', 2);
plot3 ([0, 0], [0, ry], [1, 1], 'k-', 'LineWidth', 2);
plot3 (0, 0, 1, 'ko', 'MarkerSize', 9);
text (-0.1, -0.1, 1, 'x_{c}', 'FontSize', 18);
shading flat;
view (0, 90); % reduce to x-y-plane
xlim ([-rx, rx]);
ylim ([-1, 1]);
zlim ([-rz, rz]);
axis off;
```



4.1.7 Polyhedra

A Polyhedron is an **intersection** of a finite number of **halfspaces** and **hyperplanes**.

```
x1 = [ 0.0, 0.0];
x2 = [ 0.3, 0.6];
x3 = [-0.2, 0.9];
x4 = [-0.6, 0.5];
x5 = [-0.5, 0.2];
x = [x1; x2; x3; x4; x5];
fill(x(:,1), x(:,2), ...
      [220, 220, 220] / 256, ...
      'EdgeColor', 'none');
hold on;

tprops = {'FontSize', 18};
for i = 1:5
    j = mod(i, 5) + 1;
    p = @(t) x(i,:) + t .* (x(j,:) - x(i,:));
    p0 = p(0.5); % Start of normal vector
    p1 = p(-0.2);
    p2 = p(1.2);
    % Normal vector a
    d = 0.3; % direction and scale
    if (x(i,1) < 0)
```

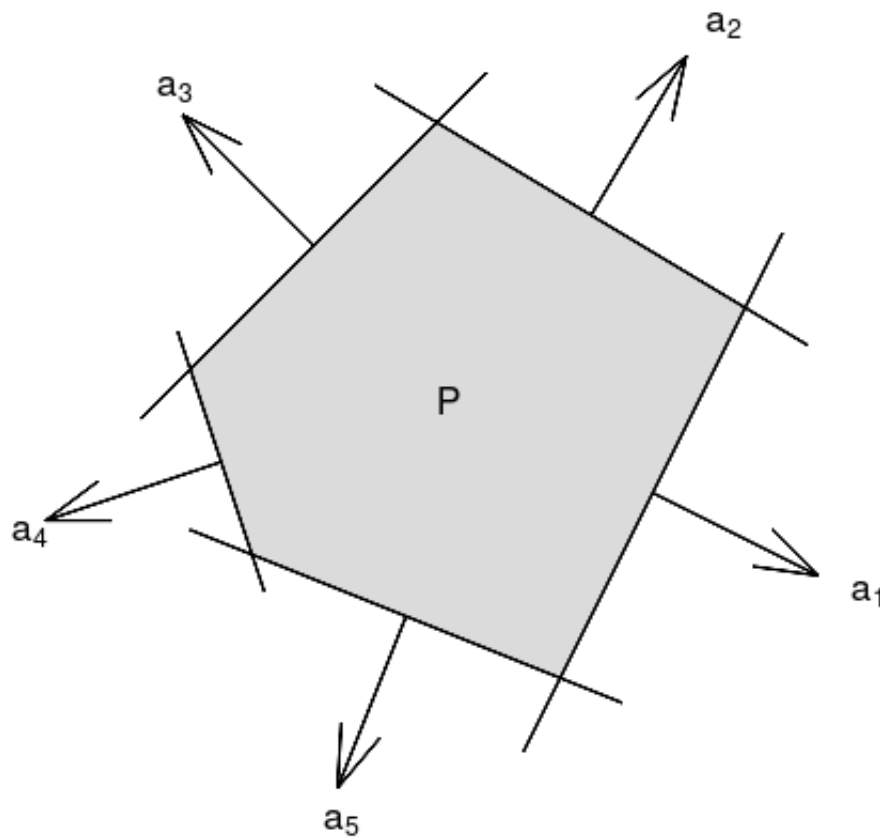
(continues on next page)

(continued from previous page)

```

    d = -d;
end
a = (x(j,:) - x(i,:))';
a = [1; -a(1)/a(2)];
a = a / norm(a) * d;
plot([p1(1), p2(1)], [p1(2), p2(2)], 'k-', 'LineWidth', 2);
quiver(p0(1), p0(2), a(1), a(2), 'k', 'LineWidth', 2);
t = p0 + 1.2 .* a;
text(t(1), t(2), sprintf('a_{%d}', i), tprops{:});
end

text(-0.2, 0.45, 'P', tprops{:});
axis equal;
axis off;
    
```



The **intersection** of convex sets is convex.

Proof.

Let $x_1, x_2 \in C \cap D$, where C and D are convex sets. Then $\theta x_1 + (1 - \theta)x_2 \in C \cap D$.

□

Solution set of finitely many linear inequalities and equalities is convex

$$Ax \leq b, \quad Cx = d,$$

where $A \in \mathbb{R}^{m \times n}$, $C \in \mathbb{R}^{p \times n}$, and \leq is component-wise inequality.

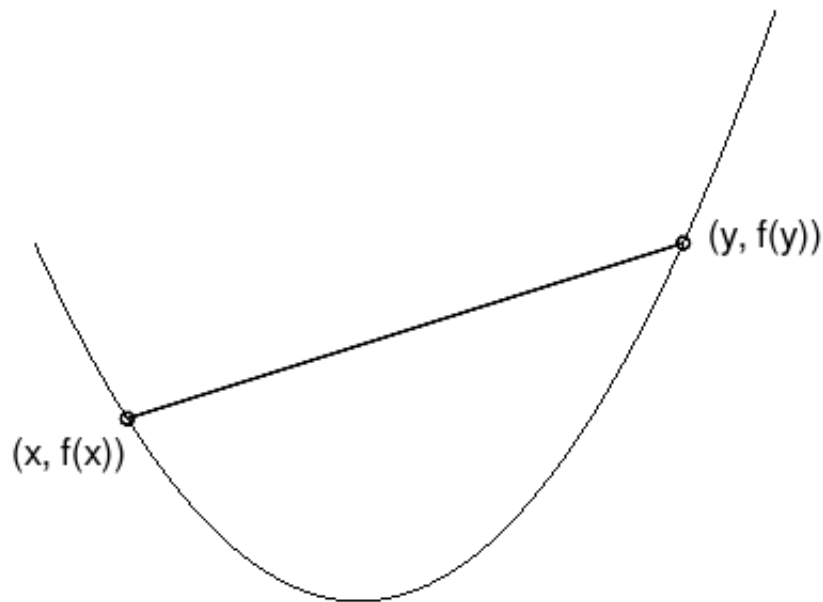
4.2 Convex functions

Let $X \subset \mathbb{R}^n$ be a convex set. A function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is **convex** on X , if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

for all $x, y \in X$ and $0 \leq \theta \leq 1$.

```
x = linspace (-0.7, 0.9, 50);
y = x.^2;
plot (x, y, 'k');
hold on;
plot ([-0.5, 0.7], [-0.5, 0.7].^2, 'ko-', 'LineWidth', 2);
text (-0.75, 0.2, '(x, f(x))', 'FontSize', 18);
text ( 0.75, 0.5, '(y, f(y))', 'FontSize', 18);
axis off;
```



- f is **concave** if $-f$ is convex.
- f is **strictly convex** if $X \subset \mathbb{R}^n$ is a convex set and

$$f(\theta x + (1 - \theta)y) < \theta f(x) + (1 - \theta)f(y),$$

for all $x, y \in X$, $x \neq y$, and $0 < \theta < 1$.

4.2.1 Examples on \mathbb{R}

Convex:

- affine: $ax + b$ on \mathbb{R} , for any $a, b \in \mathbb{R}$
- exponential: e^{ax} , for any $a \in \mathbb{R}$
- powers: x^α on \mathbb{R}_{++} , for $\alpha \geq 1$ or $\alpha \leq 0$
- powers of absolute value: $|x|^p$ on \mathbb{R} , for $p \geq 1$
- negative entropy: $x \log(x)$ on \mathbb{R}_{++}

Concave:

- affine: $ax + b$ on \mathbb{R} , for any $a, b \in \mathbb{R}$
- powers: x^α on \mathbb{R}_{++} , for $0 \leq \alpha \leq 1$
- logarithm: $\log(x)$ on \mathbb{R}_{++}

4.2.2 Examples on \mathbb{R}^n (vectors)

- affine function $f(x) = a^T x + b$ are convex and concave
- norms are convex: $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$ for $p \geq 1$; $\|x\|_\infty = \max(|x_1|, \dots, |x_n|)$

4.2.3 Examples on $\mathbb{R}^{m \times n}$ ($m \times n$ matrices)

- affine function

$$f(X) = \text{tr}(A^T X) + b = \left(\sum_{i=1}^m \sum_{j=1}^n A_{ij} X_{ij} \right) + b$$

- spectral (maximum singular value) norm

$$f(X) = \|X\|_2 = \sigma_{\max}(X) = \sqrt{\lambda_{\max}(X^T X)}.$$

Theorem 6: Unique global minimum

Let $X \subset \mathbb{R}^n$ be a convex set and $f: X \rightarrow \mathbb{R}$ a convex function, then each **local minimum** is a **global minimum**.

Furthermore, if $f \in C^1$ (continuously differentiable) over an open and convex set $X \subset \mathbb{R}^n$, then every stationary point is a global minimum of f over X .

Proof:

Assume x^* is a local, but no global minimum. Then there is $y \in X$ such that $f(y) < f(x^*)$. For $0 < \theta \leq 1$ it follows

$$f(x^* + \theta(y - x^*)) \leq f(x^*) + \theta(f(y) - f(x^*)) < f(x^*) + \theta(f(x^*) - f(x^*)) = f(x^*)$$

Thus in every neighborhood of x^* there exist points with smaller objective function value than $f(x^*)$. Therefore x^* cannot be a local minimum, contradiction.

Regarding the second statement, for $y \in X$ assume the helper function

$$\Psi(\theta) := f(x^*) + \theta(f(y) - f(x^*)) - f(x^* + \theta(y - x^*)) \geq 0.$$

Ψ is continuously differentiable and for $0 \leq \theta \leq 1$ not negative. Furthermore $\Psi(0) = 0$. Therefore

$$\Psi'(0) = (f(y) - f(x^*)) - \nabla f(x^*)^T(y - x^*) \geq 0.$$

If $\nabla f(x^*) = 0$, then $f(y) - f(x^*) \geq 0$ for all $y \in X$. Thus x^* is a global minimum of f over X .

□

4.2.4 1st order convex function condition

Theorem 7: 1st order convex function condition

Let $X \subset \mathbb{R}^n$ be a non-empty, convex set and $f \in C^1(\mathbb{R}^n, \mathbb{R})$ is convex over X , then

$$f(y) \geq f(x) + \nabla f(x)^T(y - x)$$

for all $x, y \in X$.

Proof:

1. Assume f is convex over X . For $y \in X$, $0 \leq \theta \leq 1$ the function

$$\Psi(\theta) := f(x) + \theta(f(y) - f(x)) - f(x + \theta(y - x)) \geq 0$$

is continuously differentiable and $\Psi(0) = 0$. Therefore

$$\Psi'(0) = (f(y) - f(x)) - \nabla f(x)^T(y - x) \geq 0.$$

2. Assume $f(y) \geq f(x) + \nabla f(x)^T(y - x)$ holds. Using $\bar{x} := x + \theta(y - x)$

$$f(x) \geq f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x})$$

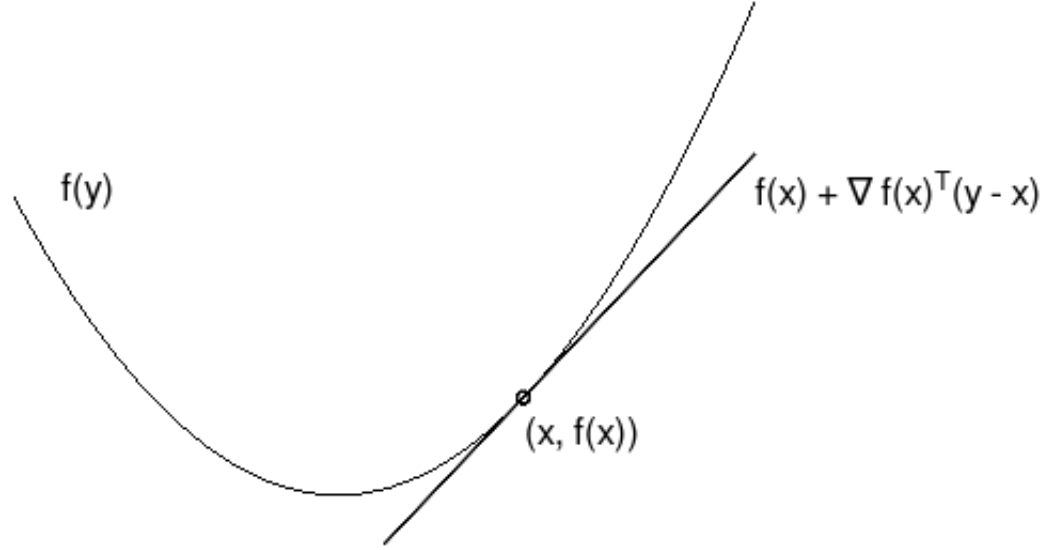
$$f(y) \geq f(\bar{x}) + \nabla f(\bar{x})^T(y - \bar{x})$$

Multiply the first inequality with $(1 - \theta)$, the second inequality with θ , and finally add both inequalities, then the assertion follows:

$$(1 - \theta)f(x) + \theta f(y) \geq f(\bar{x}) + 0 = f(x + \theta(y - x)).$$

□

```
x = linspace (-0.7, 0.9, 50);
y = x.^2;
plot (x, y, 'k');
hold on;
plot ([0.1, 0.9], 0.8 .* [0.1, 0.9] - 0.16, 'k-', 'LineWidth', 2);
plot (0.4, 0.4^2, 'ko-', 'LineWidth', 2);
text (-0.6, 0.5, 'f(y)', 'FontSize', 18);
text ( 0.4, 0.1, '(x, f(x))', 'FontSize', 18);
text ( 0.9, 0.5, 'f(x) + \nabla f(x)^T(y - x)', 'FontSize', 18);
xlim ([-0.7, 1.3]);
axis off;
```



The first-order approximation of f is **global underestimator**.

Theorem 7 can be extended to **strict convexity**, which is omitted here.

4.2.5 2nd order convex function condition

Theorem 8: 2nd order convex function condition

Let $X \subset \mathbb{R}^n$ be a open, non-empty, convex set and $f \in C^2(\mathbb{R}^n, \mathbb{R})$ is convex over X , then

$$\nabla^2 f(x) \succeq 0 \quad (\text{positive semi-definite})$$

for all $x \in X$.

Proof:

1. Assume f is convex over X . According to [Theorem 7](#) and Taylor' s Theorem for $x \in X$, $d \in \mathbb{R}^n$, and sufficiently small $t > 0$ there is

$$0 \leq f(x + td) - f(x) - t \nabla f(x)^T d = \frac{t^2}{2} d^T \nabla^2 f(x) d + o(t^2).$$

Division by $\frac{t^2}{2}$ and taking the limit $t \rightarrow 0$ results in $d^T \nabla^2 f(x) d \geq 0$, i.e. the Hessian matrix must be positive semi-definite. Note that X is an open set. Thus $x + td \in X$ holds for all sufficiently small $t > 0$.

2. Assume $\nabla^2 f(x) \succeq 0$ for all $x \in X$. According to Taylor's Theorem there is for $x, y \in X$:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x + \theta(y - x))(y - x)$$

with $0 < \theta < 1$ depending on x and y . Because the Hessian matrix is positive semi-definite

$$f(y) \geq f(x) + \nabla f(x)^T(y - x)$$

holds and therefore the convexity of f over X according to *Theorem 7*.

□

Again, the extension of Theorem 8 to **strict convexity** is omitted here.

4.2.6 Examples convex function condition

- **Quadratic function:** $f(x) = \frac{1}{2}x^T A x + b^T x + c$, where A is a symmetric $n \times n$ matrix, $x, b \in \mathbb{R}^n$, and $c \in \mathbb{R}$.

$$\nabla f(x) = Ax + b \quad \text{and} \quad \nabla^2 f(x) = A.$$

If $A \succeq 0$, then f is a convex function.

- **Least-squares objective function:** $f(x) = \|Ax - b\|_2^2$, with $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$.

$$\nabla f(x) = 2A^T(Ax - b) \quad \text{and} \quad \nabla^2 f(x) = 2A^T A$$

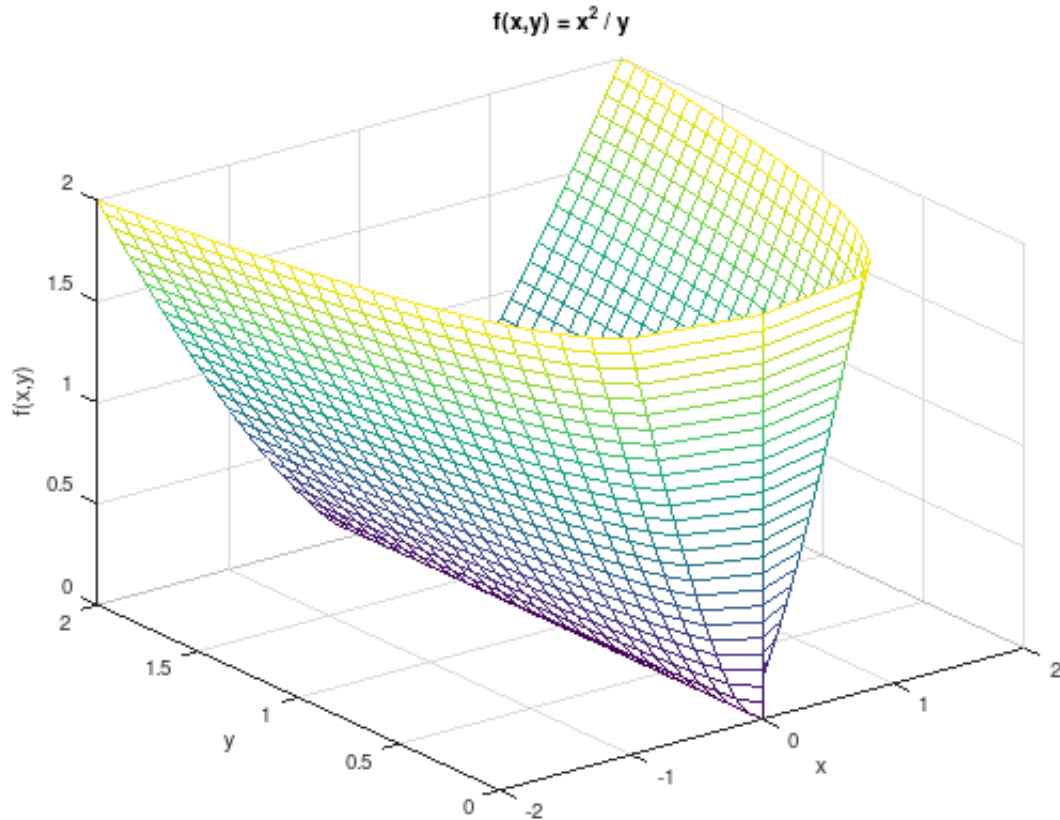
is a convex function for **any** A !

- **Quadratic-over-linear:** $f(x, y) = x^2/y$.

$$\nabla^2 f(x, y) = \frac{2}{y^3} \begin{pmatrix} y \\ -x \end{pmatrix} \begin{pmatrix} y \\ -x \end{pmatrix}^T \succeq 0$$

is a convex function for $y > 0$.

```
% Following equivalent to (x,y,z := x^2/y) for plotting.
[y, z] = meshgrid (0:.08:2);
x = sqrt(y.*z);
f = @(y, z) sqrt(x .* z);
mesh (f(y,z), y, z);
hold on;
mesh (-f(y,z), y, z);
grid on;
xlabel ('x');
ylabel ('y');
zlabel ('f(x,y)');
title ('f(x,y) = x^2 / y');
```



4.2.7 Operations that preserve convexity

- **Non-negative multiple:** αf is convex, if f is convex and $\alpha \geq 0$.
- **Sum:** $f_1 + f_2$ is convex, if f_1 and f_2 are convex. This extends to infinite sums and integrals.
- **Composition with affine function:** $f(Ax + b)$ is convex, if f is convex.
- **Point-wise maximum:** $f(x) = \max\{f_1(x), \dots, f_m(x)\}$ is convex, if f_1, \dots, f_m are convex.

More examples for convex functions:

- (Any) **norm of an affine function:** $f(x) = \|Ax + b\|$.
- **Log-barrier-function** for expressing linear inequalities:

$$f(x) = -\sum_{i=1}^m \log(b_i - a_i^T x), \quad \text{dom}(f) = \{x : a_i^T x < b_i, i = 1, \dots, m\}.$$

- **Piecewise-linear-function:** $f(x) = \max_{i=1, \dots, m} (b_i - a_i^T x)$.

For many more examples, see [BV04] (chapters 2 and 3).

CONSTRAINED MINIMIZATION PROBLEMS

Considering the general optimization problem again

$$\begin{aligned} & \text{minimize} && f(x) \\ & \text{subject to} && g_i(x) \leq 0, \quad i = 1, \dots, m, \\ & && h_j(x) = 0, \quad j = 1, \dots, p, \end{aligned} \tag{5.1}$$

with **optimization variable** $x \in \mathbb{R}^n$ and twice continuously differentiable **objective and constraint functions** $f, g_i, h_j \in C^2(\mathbb{R}^n, \mathbb{R})$.

The set of **feasible points**

$$X = \{x \in \mathbb{R}^n : g_i(x) \leq 0 \wedge h_j(x) = 0\},$$

with $g := (g_1, \dots, g_m)^T$, $h := (h_1, \dots, h_p)^T$ is assumed to be **not empty**.

For a feasible point $x \in X$ the set of **active inequalities** is denoted by

$$I(x) := \{i \in \{1, \dots, m\} : g_i(x) = 0\}.$$

For a **local minimum** $x^* \in X$ the active inequalities can be treated as equations, the inactive ones don't matter.

5.1 The Lagrangian

The key idea is to **augment** the objective function $f(x)$ with a weighted sum of the constraint functions.

The **Lagrangian** $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$ to the optimization problem (5.1) is defined as:

$$L(x, \lambda, \mu) := f(x) + \sum_{i=1}^m \lambda_i g_i(x) + \sum_{j=1}^p \mu_j h_j(x)$$

The **Lagrange multipliers** $\lambda := (\lambda_1, \dots, \lambda_m)^T$ and $\mu := (\mu_1, \dots, \mu_p)^T$ are associated with the i -th and j -th constraint function, respectively.

5.2 KKT optimality conditions

Having the Lagrangian defined, similar to the unconstrained optimization problem, first- and higher-order optimality conditions can be defined.

The first-order conditions are called **Karush-Kuhn-Tucker (KKT)** optimality conditions.

Theorem 9: KKT optimality conditions

For a local minimum x^* of the optimization problem (5.1) let the gradients

$$\{\nabla g_i(x^*), \nabla h_j(x^*) : i \in I(x^*), j = 1, \dots, p\}$$

be linear independent (**LICQ** = Linear independence constraint qualification).

Then there exist **unique** Lagrange multipliers $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^p$, such that for the Lagrangian

$$\begin{aligned} L(x^*, \lambda^*, \mu^*) &:= f(x^*) + \sum_{i=1}^m \lambda_i^* g_i(x^*) + \sum_{j=1}^p \mu_j^* h_j(x^*) \\ &= f(x^*) + (\lambda^*)^T g(x^*) + (\mu^*)^T h(x^*) \end{aligned}$$

the following necessary conditions hold:

(1) Stationarity

$$\begin{aligned} \nabla_x L(x^*, \lambda^*, \mu^*) &= \nabla f(x^*) + \sum_{i=1}^m \lambda_i^* \nabla g_i(x^*) + \sum_{j=1}^p \mu_j^* \nabla h_j(x^*) \\ &= \nabla f(x^*) + (\lambda^*)^T \nabla g(x^*) + (\mu^*)^T \nabla h(x^*) = 0 \end{aligned}$$

(2) Complementary slackness

$$(\lambda^*)^T \nabla_\lambda L(x^*, \lambda^*, \mu^*) = \sum_{i=1}^m \lambda_i^* g_i(x^*) = (\lambda^*)^T g(x^*) = 0$$

(3) Primal feasibility

$$\begin{aligned} \nabla_\mu L(x^*, \lambda^*, \mu^*) &= (h_j(x^*)) = 0, \quad j = 1, \dots, p \\ \nabla_\lambda L(x^*, \lambda^*, \mu^*) &= (g_i(x^*)) \leq 0, \quad i = 1, \dots, m \end{aligned}$$

(4) Dual feasibility

$$\lambda_i^* \geq 0, \quad i = 1, \dots, m$$

Points (x^*, λ^*, μ^*) fulfilling the KKT optimality conditions are referred to as **KKT-points**.

Study the exercises [RM01](#), [RM02](#), and [RM03](#) to apply the KKT optimality conditions.

For a proof and more detailed discussion on the KKT optimality conditions, see [\[BV04\]](#) (chapter 5) and [\[LY16\]](#) (chapter 11).

Summarizing key ideas:

- The KKT optimality conditions help to find optimal solutions for constraint optimization problems by defining a **non-linear system of equations** to solve.

$$\begin{aligned} \nabla_x L &= 0 \\ \nabla_\lambda L &= 0 \\ \nabla_\mu L &= 0 \end{aligned}$$

Many **solution algorithms** for constraint optimization problems solve the KKT optimality conditions numerically.

- KKT-points (x^*, λ^*) are **saddle-points** of the Lagrangian.

- Approach to find a supporting hyperplane on the feasible set of inequality constraints

$$\{x \in X: g_i(x) \leq 0, \quad i = 1, \dots, m\}.$$

The proof of the KKT optimality conditions makes use of the hyperplane separation theorem.

- From the Lagrangian and the KKT optimality conditions to each optimization problem (5.1) a **convex dual optimization problem** can be formulated. See [BV04] (chapter 5).

CONSTRAINED MINIMIZATION ALGORITHMS

Some of the algorithms and techniques used to find local minima of **unconstrained** optimization problems can be applied *mutatis mutandis* for the **constrained** problem case.

6.1 Sequential Quadratic Programming (SQP)

For the sake of simplicity the equality constrained minimization problem is considered:

$$\begin{aligned} & \text{minimize} && f(x) \\ & \text{subject to} && h_j(x) = 0, \quad j = 1, \dots, p, \end{aligned}$$

with **optimization variable** $x \in \mathbb{R}^n$ and twice continuously differentiable **objective and constraint functions** $f, h_j \in C^2(\mathbb{R}^n, \mathbb{R})$, $h := (h_1, \dots, h_p)^T \in C^2(\mathbb{R}^n, \mathbb{R}^p)$.

The corresponding **Lagrangian** is:

$$L(x, \mu) := f(x) + \sum_{j=1}^p \mu_j h_j(x),$$

and **KKT optimality conditions**:

$$\begin{aligned} \nabla_x L(x^*, \mu^*) &= \nabla f(x^*) + \sum_{j=1}^p \mu_j^* \nabla h_j(x^*) = 0 \\ \nabla_\mu L(x^*, \mu^*) &= (h_j(x^*)) = h(x^*) = 0, \quad j = 1, \dots, p \end{aligned}$$

Again the goal is to perform an iteration

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

with $\mathbf{x} = (x, \mu)^T$, which converges from a starting point \mathbf{x}^0 to a local minimum \mathbf{x}^* (a stationary point of the KKT optimality conditions) choosing a **descent direction** $\mathbf{d} = (\Delta x, \Delta \mu)^T$ and a **step size** α in each step (cf. [Gradient methods](#)).

Using [Newton's method](#) (quadratic Taylor-approximation of the Lagrangian $L(x, \mu)$) to determine the descent direction, results in solving the following system of non-linear equations:

$$\nabla^2 L(x, \mu) \mathbf{d} = -\nabla L(x, \mu)$$

$$\begin{pmatrix} \nabla_{x,x}^2 L(x, \mu) & \nabla_{x,\mu}^2 L(x, \mu) \\ \nabla_{\mu,x}^2 L(x, \mu) & \nabla_{\mu,\mu}^2 L(x, \mu) \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta \mu \end{pmatrix} = - \begin{pmatrix} \nabla_x L(x, \mu) \\ \nabla_\mu L(x, \mu) \end{pmatrix}$$

This **Hessian matrix** has a certain structure:

$$\begin{pmatrix} Q & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta \mu \end{pmatrix} = - \begin{pmatrix} \nabla f(x) + B^T \mu \\ h(x) \end{pmatrix} = - \begin{pmatrix} \nabla_x L(x, \mu) \\ h(x) \end{pmatrix},$$

where:

- $Q := \nabla_{x,x}^2 L(x, \mu) = \nabla^2 f(x) + \sum_{j=1}^p \mu_j \nabla^2 h_j(x)$.
- $B := \nabla_{\mu,x}^2 L(x, \mu) = \begin{pmatrix} \nabla^T h_1(x) \\ \vdots \\ \nabla^T h_p(x) \end{pmatrix}$ is the **Jacobian matrix** of $h(x)$.
- $B^T = \nabla_{x,\mu}^2 L(x, \mu) = (\nabla h_1(x) \quad \dots \quad \nabla h_p(x))$ is the transposed **Jacobian matrix** of $h(x)$.
- $\nabla_{\mu,\mu}^2 L(x, \mu) = 0$ is a zero matrix, as $L(x, \mu)$ is linear in μ .

The derived Newton-equation happens to be identical to the KKT optimality conditions of the following quadratic optimization problem:

$$\begin{aligned} \text{minimize} \quad & F(\Delta x) := \frac{1}{2}(\Delta x)^T Q \Delta x + \nabla_x L(x, \mu)^T \Delta x \\ \text{subject to} \quad & H(\Delta x) := B \Delta x + h(x) = 0. \end{aligned} \tag{6.1}$$

By choosing the Lagrange multipliers $\lambda = \Delta \mu$ for the equality constraints, the corresponding Lagrangian is:

$$\mathcal{L}(\Delta x, \lambda) = F(\Delta x) + H(\Delta x)^T \lambda$$

and the KKT optimality conditions:

$$\begin{aligned} \nabla_{(\Delta x)} \mathcal{L}(\Delta x, \lambda) &:= Q \Delta x + \nabla_x L(x, \mu) + B^T \lambda = 0, \\ H(\Delta x) = \nabla_{\lambda} \mathcal{L}(\Delta x, \lambda) &:= B \Delta x + h(x) = 0. \end{aligned}$$

or written in matrix notation and regarding $\lambda = \Delta \mu$:

$$\begin{pmatrix} Q & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta \mu \end{pmatrix} = - \begin{pmatrix} \nabla_x L(x, \mu) \\ h(x) \end{pmatrix}.$$

6.1.1 Summarizing

- SQP methods solve a sequence of **quadratic optimization sub-problems** (quadratic objective function and linearized constraints).
- In the unconstrained case, SQP reduces to **Newton's method** for finding a stationary point $\nabla f(x) = 0$.
- In the only equality constrained case, the SQP method is equivalent to applying Newton's method to the KKT optimality conditions.
- Extensions for equality and inequality constraints exist, but are not discussed in this seminar.
- Similar problems and limitations as discussed in the section about **Newton's method** apply to the SQP method.
 - Computing the Hessian matrix is expensive.
 - If the Hessian matrix is not symmetric positive definite or the constraints are not linear independent (LICQ) during the iteration, convergence cannot be guaranteed.
- See exercise [RM04](#) for an application example.

INTERIOR-POINT METHODS (IPM)

Penalty-, Barrier-, and Interior-Point Methods are a group of methods that solve constrained optimization problems iteratively by solving a sequence of unconstrained optimization problems and penalizing the violation of constraints:

- **Penalty Methods:** impose a penalty for violating a constraint
- **Barrier methods, IPM:** impose a penalty for reaching the boundary of an inequality constraint

Idea: Suppose that the constrained optimization problem has the form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in X. \end{array}$$

Define $\mathfrak{b}(x) = 0$ for $x \in X$ and $\mathfrak{b}(x) = +\infty$ otherwise (**penalty**). Then the constrained problem is equivalent to the unconstrained problem

$$\text{minimize} \quad f(x) + \mathfrak{b}(x)$$

for all $x \in \mathbb{R}^n$.

Consider the non-linear inequality-constrained optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_j(x) \geq 0, \quad j = 1, \dots, m. \end{array}$$

IPMs maintain feasibility by creating a barrier function $\beta(x, \mu)$ keeping the iterates $x(\mu^k)$ for decreasing $\mu = \mu^k \rightarrow 0$ away from the boundary of the feasible region:

$$\begin{array}{ll} \text{minimize} & \beta(x, \mu) := f(x) - \mu \sum_{j=1}^m \log(g_j(x)) \\ \text{subject to} & g_j(x) > 0, \quad j = 1, \dots, m, \\ & \mu > 0. \end{array}$$

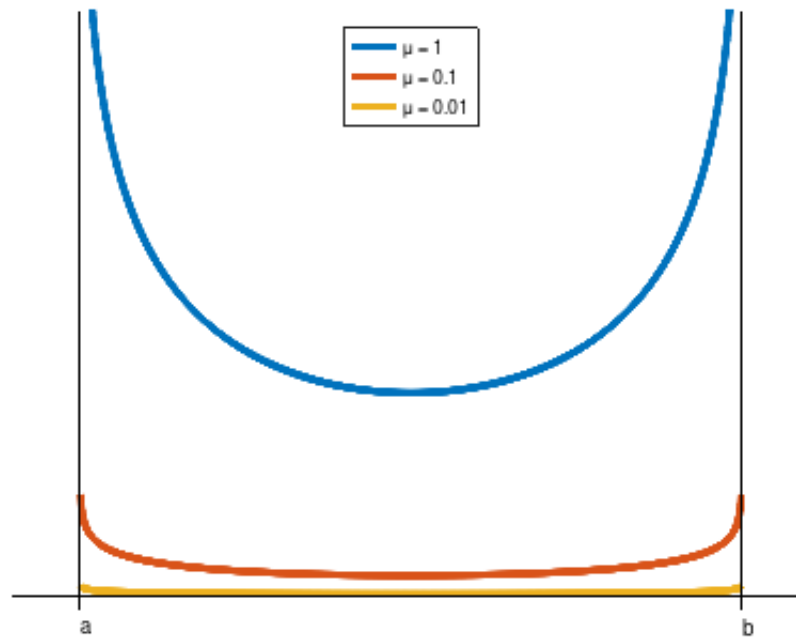
Example: $\beta(x, \mu) := 0 - \mu (\log(x - a) + \log(b - x))$, notice that $f(x) \equiv 0$ implies that each feasible point $x \in [a, b]$ is optimal.

```
a = 1;
b = 2;
beta = @(x,mu) 0 - mu * (log(x - a) + log(b - x));
mu = [1; 0.1; 0.01];
x = linspace(a, b, 1000);
y = beta(x, mu);
plot(x, y, 'LineWidth', 4);
hold on;
plot([a - 0.1, b + 0.1], [0, 0], 'k');
```

(continues on next page)

(continued from previous page)

```
plot ([a, a], [-0.1, 4], 'k');
plot ([b, b], [-0.1, 4], 'k');
ylim ([-1 4]);
legend (strcat ({'\mu = '}, ...
    cellfun (@num2str, num2cell (mu), ...
        'UniformOutput', false)), ...
    'Location', 'North');
text (a, -0.2, 'a');
text (b, -0.2, 'b');
axis off;
```



7.1 IPM: Example 1 - Barrier Method

Consider the non-linear optimization problem:

$$\begin{aligned} & \text{minimize} && f(x_1, x_2) = x_1 - 2x_2 \\ & \text{subject to} && g_1(x_1, x_2) = 1 + x_1 - x_2^2 \geq 0, \\ & && g_2(x_1, x_2) = x_2 \geq 0. \end{aligned}$$

```
% Optimal point.
px = 0;
py = 1;
```

(continues on next page)

(continued from previous page)

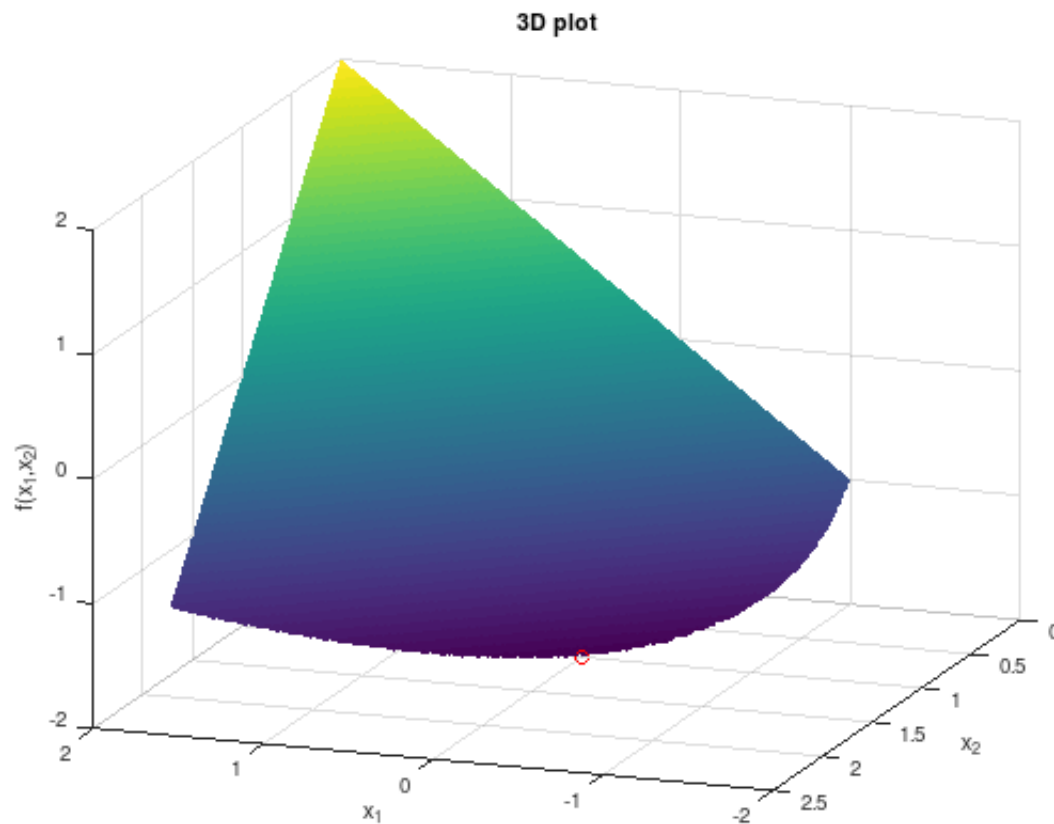
```

pz = -2;

[X1, X2] = meshgrid (linspace (-2, 2, 500),linspace (0, 2.5, 500));
FX = X1 - 2*X2;

% Remove infeasible points.
FX((1 + X1 - X2.^2) < 0) = inf;
surf (X1, X2, FX);
shading flat;
hold on;
plot3 (px, py, pz, 'ro');
xlabel ('x_1');
ylabel ('x_2');
zlabel ('f(x_1,x_2)');
title ('3D plot');
view (200, 20);

```



Then the logarithmic barrier function gives the unconstrained problem

$$\min_x \beta(x, \mu) := x_1 - 2x_2 - \mu \log(1 + x_1 - x_2^2) - \mu \log(x_2)$$

for a sequence of decreasing barrier parameters. For a specific parameter μ , the first-order necessary conditions for

optimality are:

$$\begin{aligned} 1 - \frac{\mu}{1 + x_1 - x_2^2} &= 0, \\ -2 - \frac{2\mu x_2}{1 + x_1 - x_2^2} - \frac{\mu}{x_2} &= 0. \end{aligned}$$

If the constraints are strictly satisfied, the denominators are positive. An equation for x_2 can be derived:

$$x_2^2 - x_2 - \frac{1}{2}\mu = 0.$$

This equation can be solved to determine x_2 and x_1 in terms of μ :

$$\begin{aligned} x_1(\mu) &= \frac{\sqrt{1 + 2\mu} + 3\mu - 1}{2}, \\ x_2(\mu) &= \frac{\sqrt{1 + 2\mu} + 1}{2}. \end{aligned}$$

The unconstrained objective is strictly convex and hence this solution is the unique local minimizer in the feasible region. As μ approaches zero

$$\begin{aligned} \lim_{\mu \rightarrow 0_+} x_1(\mu) &= \frac{\sqrt{1 + 2\mu} + 3\mu - 1}{2} = 0, \\ \lim_{\mu \rightarrow 0_+} x_2(\mu) &= \frac{\sqrt{1 + 2\mu} + 1}{2} = 1. \end{aligned}$$

One can verify that $x^* = (0, 1)^T$ is indeed the solution to this optimization problem.

```
x1 = @(mu) (sqrt(1 + 2 * mu) + 3 * mu - 1) / 2;
x2 = @(mu) (sqrt(1 + 2 * mu) + 1) / 2;
mu = 10.^(0:-1:-6);
x1 = x1(mu);
x2 = x2(mu);
disp(' mu          x1(mu)          x2(mu) ');
disp('-----')
for i = 1:length(mu)
    fprintf('10^(%2d)    %.7f    %.7f\n', -i + 1, x1(i), x2(i));
end
```

mu	x1(mu)	x2(mu)
10^(0)	1.8660254	1.3660254
10^(-1)	0.1977226	1.0477226
10^(-2)	0.0199752	1.0049752
10^(-3)	0.0019998	1.0004998
10^(-4)	0.0002000	1.0000500
10^(-5)	0.0000200	1.0000050
10^(-6)	0.0000020	1.0000005

This example demonstrates a number of features typical for IPMs:

- If $f(x)$ is convex, then $\beta(x, \mu)$ is strictly convex, since the barrier function is strictly convex, and the minimizer $x(\mu)$ is unique.
- **Central path** $\{x(\mu) := \arg \min_x \beta(x, \mu) : \mu > 0\}$. IPMs that maintain close to the central path are called **path-following IPMs**. Under mild conditions **convergence** to the optimal solution can be proved.
- Disadvantage: **ill-conditioning** of the Hessian matrix of $\beta(x, \mu)$ for small $\mu > 0$.

7.2 IPM: Example 2 - Ill-conditioned Hessian matrix

Consider the non-linear optimization problem:

$$\begin{aligned} & \text{minimize} && f(x) = x_1^2 + x_2^2 \\ & \text{subject to} && g_1(x) = x_1 - 1 \geq 0, \\ & && g_2(x) = x_2 + 1 \geq 0. \end{aligned}$$

The solution to this problem is $x^* = (1, 0)^T$. The first inequality is active at x^* , and the corresponding Lagrange multiplier is $\lambda_1^* = 2$. The second constraint is inactive; hence its Lagrange multiplier is $\lambda_2^* = 0$.

Try to show this using the *KKT optimality conditions*.

The Lagrangian is

$$L(x, \lambda) := x_1^2 + x_2^2 - \lambda_1(x_1 - 1) - \lambda_2(x_2 + 1)$$

and respective the stationary conditions are

$$\nabla_{x_1} L(x, \lambda) = 2x_1 - \lambda_1 = 0,$$

$$\nabla_{x_2} L(x, \lambda) = 2x_2 - \lambda_2 = 0.$$

Suppose the problem is solved via a logarithmic barrier function using an unconstrained minimization solver such as Newton's method:

$$\min_x \beta(x, \mu) := x_1^2 + x_2^2 - \mu \log(x_1 - 1) - \mu \log(x_2 + 1)$$

for a decreasing sequence of barrier parameters μ that converge to zero. The first-order necessary conditions for optimality $\nabla_x \beta(x, \mu) = 0$ are:

$$2x_1 - \underbrace{\frac{\mu}{x_1 - 1}}_{\text{KKT } \lambda_1} = 0,$$

$$2x_2 - \underbrace{\frac{\mu}{x_2 + 1}}_{\text{KKT } \lambda_2} = 0,$$

yielding the unconstrained minimizers

$$\begin{aligned} x_1(\mu) &= \frac{1 + \sqrt{1 + 2\mu}}{2}, \\ x_2(\mu) &= \frac{-1 + \sqrt{1 + 2\mu}}{2}. \end{aligned}$$

From the KKT stationary conditions for small $\mu > 0$ one obtains

$$\lambda_1(\mu) := 2x_1(\mu) \approx 2,$$

$$\lambda_2(\mu) := 2x_2(\mu) \approx 0.$$

Since

$$\nabla_{x,x}^2 \beta(x, \mu) = \begin{pmatrix} 2 + \frac{\mu}{(x_1 - 1)^2} & 0 \\ 0 & 2 + \frac{\mu}{(x_2 + 1)^2} \end{pmatrix}$$

for small μ

$$\nabla_{x,x}^2 \beta(x, \mu) = \begin{pmatrix} 2 + \frac{\lambda_1^2(\mu)}{\mu} & 0 \\ 0 & 2 + \frac{\lambda_2^2(\mu)}{\mu} \end{pmatrix} \approx \begin{pmatrix} 2 + \frac{4}{\mu} & 0 \\ 0 & 2 \end{pmatrix}$$

The condition number of the Hessian matrix is approximately equal to

$$\frac{2 + \frac{4}{\mu}}{2} = 1 + \frac{2}{\mu} = O\left(\frac{1}{\mu}\right).$$

Hence the matrix is ill-conditioned. Although the calculations were performed for a point on the barrier trajectory, the same results will hold at all points in a neighborhood of the solution.

The ill-conditioning of the Hessian rules out unconstrained methods such as *Newton's method*, trust region methods etc., and led to their abandonment.

To get the idea right, consider the following two plots of the barrier function $\beta(x, \mu)$ for $\mu = 1$ and $\mu = 0.001$.

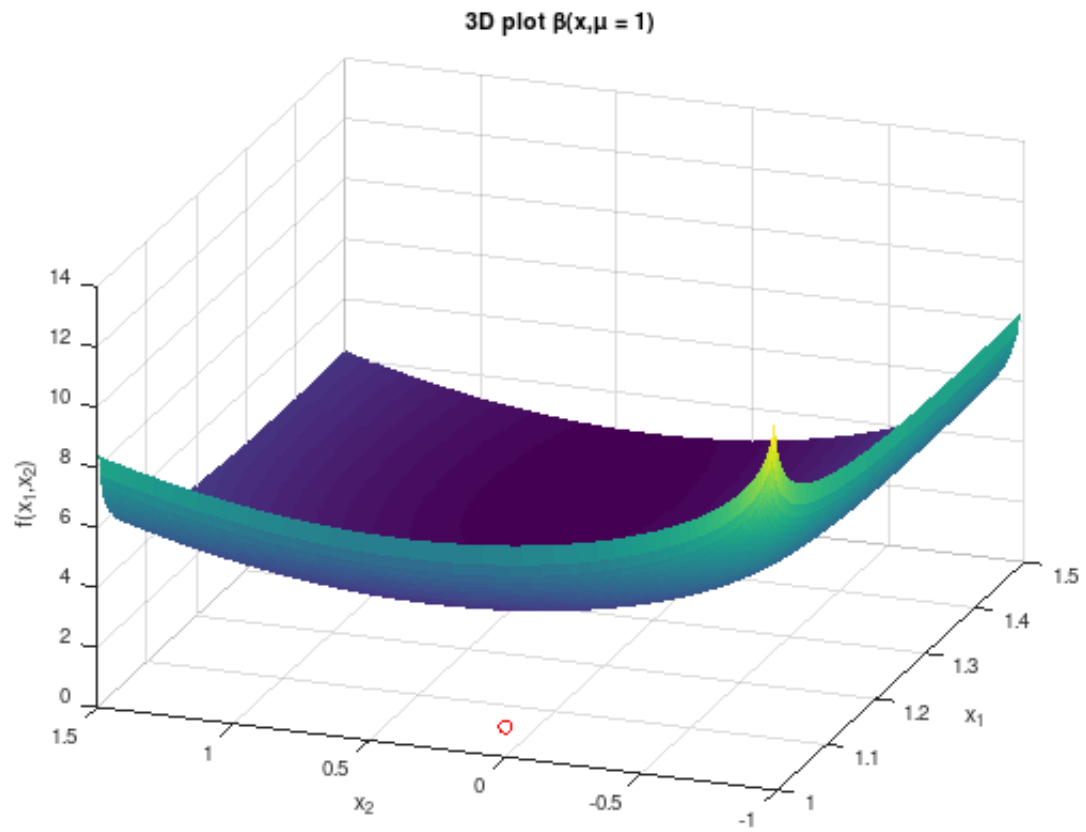
```
% Optimal point.
px = 1;
py = 0;
pz = 1;

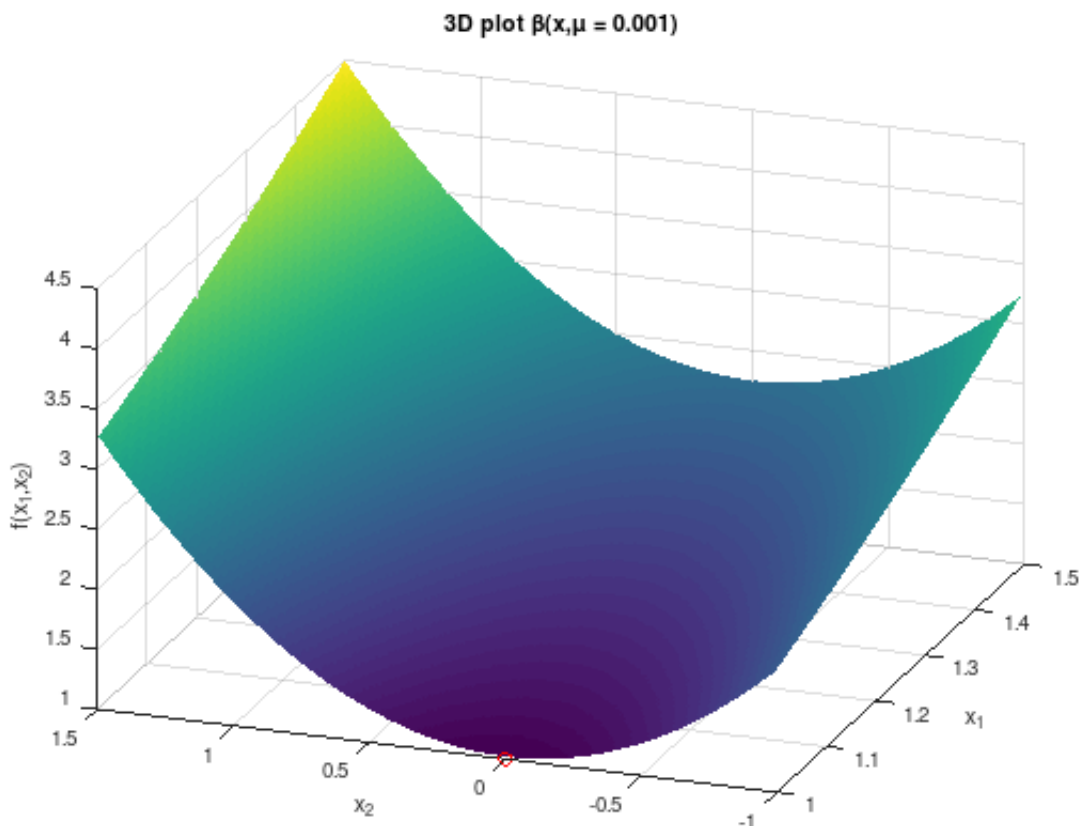
[X1, X2] = meshgrid (linspace (1, 1.5, 200), linspace (-1, 1.5, 200));
FX = @(mu) X1.^2 + X2.^2 - mu .* log( X1 - 1 ) - mu .* log( X2 + 1 );

surf (X1, X2, FX(1));
shading flat;
hold on;
plot3 (px, py, pz, 'ro');
xlabel ('x_1');
ylabel ('x_2');
zlabel ('f(x_1,x_2)');
title ('3D plot \beta(x, \mu = 1)');
view (-70, 30);

figure

surf (X1, X2, FX(0.001));
shading flat;
hold on;
plot3 (px, py, pz, 'ro');
xlabel ('x_1');
ylabel ('x_2');
zlabel ('f(x_1,x_2)');
title ('3D plot \beta(x, \mu = 0.001)');
view (-70, 30);
```





As seen previous exercises and the theory for unconstrained optimization problems, the second figure has an “ideal shape” for applying Newton’s method, compared to the first figure. The minimum (red circle) is clearly at “the bottom” of the bowl-shaped function. However, for applying Newton’s method the Hessian matrix $\nabla_{x,x}^2 \beta(x, \mu)$ for $\mu = 0.001$ and smaller is ill-conditioned and the resulting linear system is difficult to solve.

7.3 A simple transformation

Interest was renewed in 1984 by Karmarkar’s paper [Kar84]. Consider a general optimization problem:

$$\begin{aligned} & \text{minimize} && f(x) \\ & \text{subject to} && g_i(x) \leq 0, \quad i = 1, \dots, m, \\ & && h_j(x) = 0, \quad j = 1, \dots, p, \end{aligned}$$

The optimality conditions of our barrier problem

$$\text{minimize} \quad \beta(x, \mu, y) := f(x) - \mu \sum_{i=1}^m \log(g_i(x)) + \sum_{j=1}^p y_j h_j(x)$$

with lagrange multiplier $y \in \mathbb{R}^p$ are

$$\begin{aligned}\nabla_x \beta(x, \mu, y) &= \nabla f(x) - \sum_{i=1}^m \frac{\mu}{g_i(x)} \nabla g_i(x) + \sum_{j=1}^p y_j \nabla h_j(x) \\ &= \nabla f(x) - \sum_{i=1}^m s_i \nabla g_i(x) + \sum_{j=1}^p y_j \nabla h_j(x) = 0 \\ g_i(x) &> 0, \quad i = 1, \dots, m, \\ h_j(x) &= 0, \quad j = 1, \dots, p,\end{aligned}$$

with the additional variables $s_i := \frac{\mu}{g_i(x)} > 0$ for interior points.

Hence, the **first-order optimality conditions of the barrier problem** are

$$\begin{aligned}\nabla f(x) - \sum_{i=1}^m s_i \nabla g_i(x) + \sum_{j=1}^p y_j \nabla h_j(x) &= 0, \\ h_j(x) &= 0, \\ g_i(x) &> 0, \\ s_i g_i(x) &= \mu, \\ s_i &> 0,\end{aligned}$$

for $i = 1, \dots, m$ and $j = 1, \dots, p$.

The equations can be rewritten as **non-linear system of equations**:

$$\begin{aligned}\nabla f(x) - G'(x)^T s + H'(x)^T y &= 0, \\ h(x) &= 0, \\ G(x)s - M &= 0,\end{aligned}$$

with $G(x) := \begin{pmatrix} g_1(x) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & g_m(x) \end{pmatrix}$ and Jacobian matrix $G'(x) := \begin{pmatrix} \nabla^T g_1(x) \\ \vdots \\ \nabla^T g_m(x) \end{pmatrix}$, $h(x) := \begin{pmatrix} h_1(x) \\ \vdots \\ h_p(x) \end{pmatrix}$, $H'(x) := \begin{pmatrix} \nabla^T h_1(x) \\ \vdots \\ \nabla^T h_p(x) \end{pmatrix}$, $s := \begin{pmatrix} s_1 \\ \vdots \\ s_m \end{pmatrix}$, and $M := \begin{pmatrix} \mu \\ \vdots \\ \mu \end{pmatrix}$.

Finally, one can apply **Newton's method**

$$\begin{pmatrix} x^{k+1} \\ s^{k+1} \\ y^{k+1} \end{pmatrix} = \begin{pmatrix} x^k \\ s^k \\ y^k \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta y \end{pmatrix}.$$

The **linear Newton equation** has the form (cf. [BV04], p. 610):

$$\begin{pmatrix} \nabla^2 f(x^k) - \sum_{i=1}^m s_i^k \nabla^2 g_i(x^k) + \sum_{j=1}^p y_j^k \nabla^2 h_j(x^k) & -G'(x^k)^T & H'(x^k)^T \\ \text{diag}(s^k)G'(x^k) & G(x^k) & 0 \\ H'(x^k) & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta y \end{pmatrix} = - \begin{pmatrix} \nabla f(x^k) - G'(x^k)^T s^k + H'(x^k)^T y^k \\ G(x^k)s^k - M \\ h(x^k) \end{pmatrix}.$$

- The Hessian is now well-conditioned due to the additional variables s .
- There are several modifications due to solving Newton's equations.

7.4 IPM Example 3 - Linear Program

Consider a *Linear Program* (LP):

$$\begin{aligned} \text{minimize} \quad & f(x) = c^T x \\ \text{subject to} \quad & h(x) = Ax - b = 0, \\ & g(x) = x \geq 0, \end{aligned}$$

where $c, x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$, the comparison operator \geq is element-wise.

Constructing the linear Newton equation above is straight forward for LPs:

$$\begin{pmatrix} 0 & -I & A^T \\ \text{diag}(s^k) & \text{diag}(x^k) & 0 \\ A & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta y \end{pmatrix} = - \begin{pmatrix} c - s^k + A^T y^k \\ \text{diag}(x^k) s^k - M \\ Ax^k - b \end{pmatrix},$$

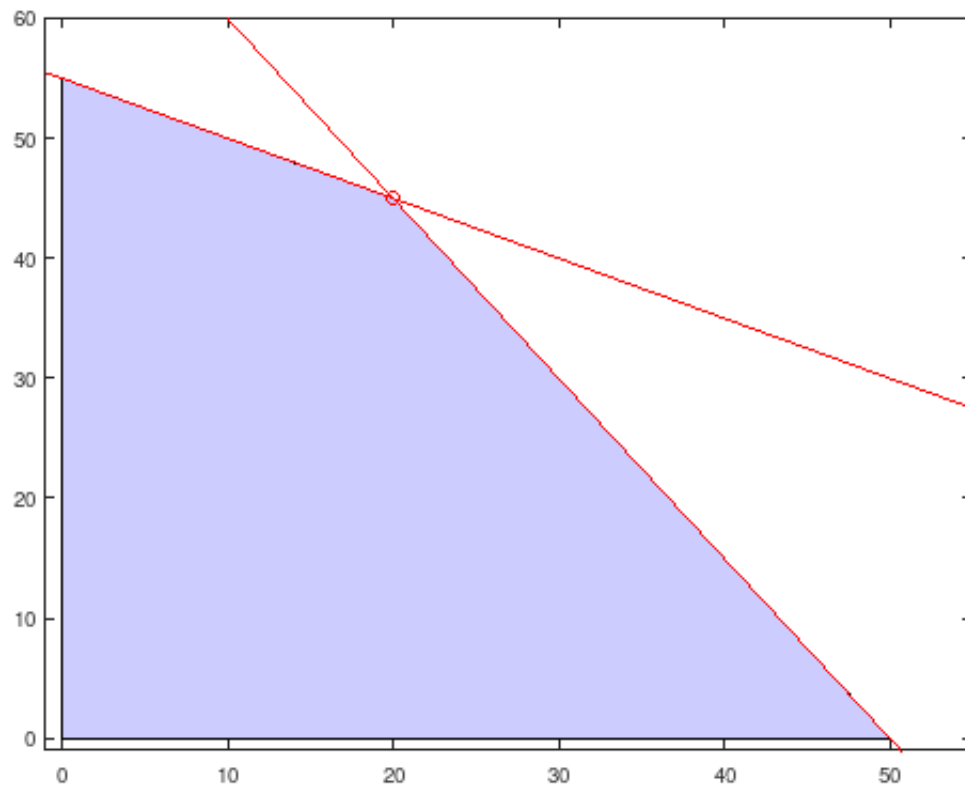
where $I \in \mathbb{R}^{n \times n}$ is the unit matrix.

Now the theory should be applied to a small LP to construct a strongly simplified IPM solution algorithm. Consider:

$$\begin{aligned} \text{minimize} \quad & \begin{pmatrix} 4 & 3 \end{pmatrix} x \\ \text{subject to} \quad & \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix} x \leq \begin{pmatrix} 220 \\ 150 \end{pmatrix}, \\ & x \geq 0, \end{aligned}$$

with optimal point $x^* = (20, 45)^T$ and optimal objective function value 215. The set of feasible points is visualized in the following figure.

```
x = linspace (-1, 55, 100);
y1 = (-2 * x + 220) / 4;
y2 = (-3 * x + 150) / 2;
fill ([0, 50, 20, 0], [0 0 45, 55], ...
      'b', 'FaceAlpha', 0.2);
hold on;
plot (x, y1, 'r');
plot (x, y2, 'r');
xlim ([-1 55]);
ylim ([-1 60]);
plot (20, 45, 'ro');
```



7.4.1 Simplified IPM algorithm for LP

The following listing sketches a very simplified IPM algorithm to solve Linear Programs. See also [lp_solver.m](#) in the appendix.

```
% LP data converted to equality constraints using slack variables.
c = [ 4 3 0 0 ]';
A = [ 2 4 -1 0; ...
      3 2 0 -1 ];
b = [ 220; 150 ];

n = length(c);
m = length(b);

% Hessian matrix for Newton step.
H = @(x,s) [ zeros(n), -eye(n), A'; ...
             diag(s), diag(x), zeros(n, m); ...
             A, zeros(m, n + m) ];

% Gradient vector for Newton step.
G = @(x,s,y,mu) [ c - s + A' * y; ...
                  x .* s - mu * ones(n, 1); ...
                  A * x - b ];
```

(continues on next page)

(continued from previous page)

```
% Initial values. x should be an inner LP point.
x = [ 10 10 0 0 ]';
s = ones (n, 1);
y = ones (m, 1);
mu = 0.1;

% Track central path.
xpath = x;

fprintf('Iteration   fval       ||dx||       ||dy||       ||ds||\n')

% Newton iteration.
% Assume 10 steps would be enough.
for i = 1:10

    % Newton step on optimality conditions.
    d = H(x,s) \ -G(x,s,y,mu);

    % Stop iteration if x does not change any more.
    if (norm (d(1:n), 'inf') < 1e-6)
        fprintf ('\nIPM converged in step %d.\n\n', i);
        break;
    end

    % Update variables.
    x = x + d(1:n);
    s = s + d((1:n) + n);
    y = y + d((1:m) + (n * 2));

    mu = mu^2; % Shrink mu fast.

    % Output step statistics.
    xpath(:,end+1) = x;
    fprintf('%5d      %.2f      %.2e      %.2e      %.2e\n', ...
        i, c'*x, ...
        norm (d(1:n), 'inf'), ...
        norm (d((1:m) + n), 'inf'), ...
        norm (d((1:m) + (n * 2)), 'inf'));
end

% Output optimal point.
x = x'
```

Iteration	fval	dx	dy	ds
1	215.14	3.50e+01	4.49e+00	2.19e+00
2	214.97	1.32e-01	3.49e+00	1.06e+00
3	215.00	9.86e-02	2.76e-03	2.06e-03
4	215.00	5.47e-04	1.06e-05	4.06e-06

IPM converged in step 5.

x =

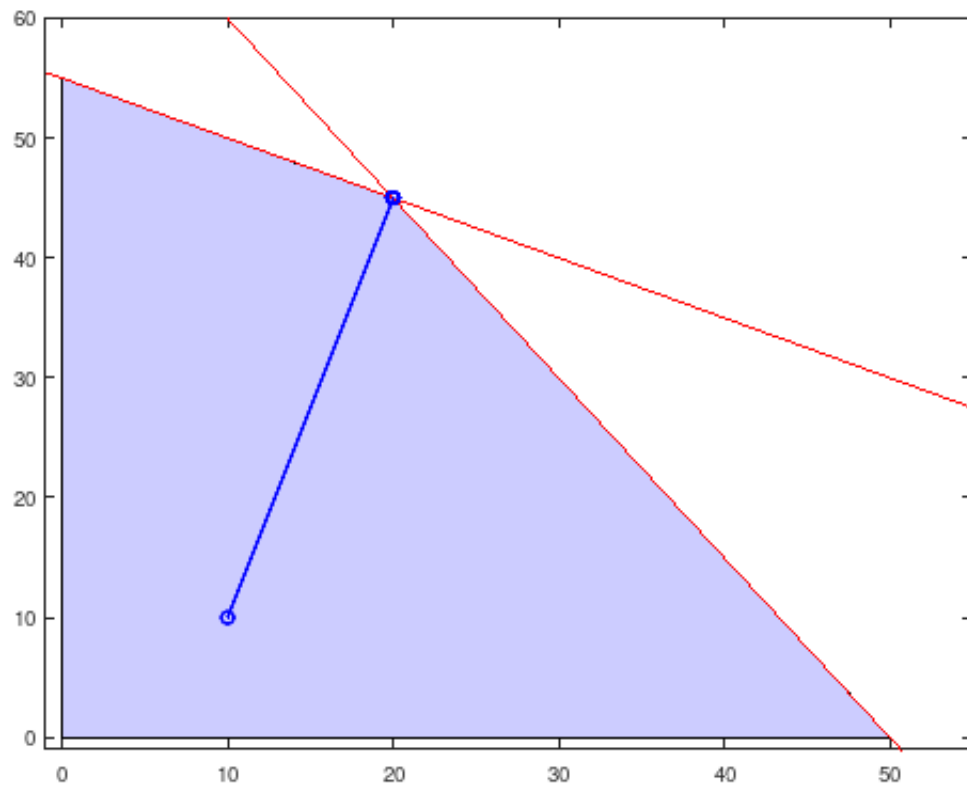
2.0000e+01	4.5000e+01	7.6510e-08	7.9108e-09
------------	------------	------------	------------

After five steps an optimal point with accuracy of 10^{-6} was found. The **central path** is shown in the following figure.


```

x = linspace (-1, 55, 100);
y1 = (-2 * x + 220) / 4;
y2 = (-3 * x + 150) / 2;
fill ([0, 50, 20, 0], [0 0 45, 55], ...
      'b', 'FaceAlpha', 0.2);
hold on;
plot (x, y1, 'r');
plot (x, y2, 'r');
xlim ([-1 55]);
ylim ([-1 60]);
plot (20, 45, 'ro');
plot (xpath(1,:), xpath(2,:), 'bo-', 'LineWidth', 2)

```



Already after the first step the simple IPM computes a point close to the optimal point and only improves the accuracy with decreasing values of μ .

7.5 Summarizing

- Interior-Point Methods (IPMs) impose a penalty for reaching the boundary of an inequality constraint.
- IPM uses a barrier function $\beta(x, \mu) := f(x) - \mu \sum_{i=1}^m \log(g_i(x))$ to achieve this goal.
- Solving the first-order optimality condition with Newton's Method is difficult due to an **ill-conditioned Hessian matrix**.
- Introducing additional variables $s_i := \frac{\mu}{g_i(x)} > 0$ improves the Hessian matrix condition, but increases the system to solve. Specialized solving techniques are required for practical applications.
- A simplified IPM for LP problems has been presented.
- Suggested reading [GNS08] (chapter 16).

PENALTY METHODS

In contrast to barrier methods, **penalty methods** solve a sequence of unconstrained optimization problems whose solution is usually infeasible to the original constrained problem. As this penalty is increased, the iterates are forced towards the feasible region.

Consider the equality-constrained problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & h(x) = 0, \end{array}$$

where $h(x)$ is an p -dimensional vector, whose j -th component $h_j(x)$ is twice continuously differentiable.

The best-known penalty is the **quadratic-loss function**

$$\psi(x) := \frac{1}{2} \sum_{j=1}^p h_j(x)^2 = \frac{1}{2} h(x)^T h(x).$$

The weight of the penalty is controlled by a positive **penalty parameter** ρ . The penalty method consists of solving a sequence of **unconstrained minimization problems** of the form

$$\min_x \pi(x, \rho^k) = f(x) + \rho^k \psi(x)$$

for an increasing sequence $\{\rho^k\}$ of positive values tending to infinity.

Penalty methods share many of the properties of barrier methods. Under appropriate conditions, the sequence of penalty function minimizers defines a continuous trajectory. In the latter case, it is possible to get estimates of the Lagrange multipliers at the solution.

Consider the quadratic-loss penalty function

$$\pi(x, \rho) = f(x) + \frac{1}{2} \rho \sum_{j=1}^p h_j(x)^2.$$

Its minimizer $x(\rho)$ satisfies

$$\nabla_x \pi(x(\rho), \rho) = \nabla f(x(\rho)) + \rho \sum_{j=1}^p \nabla h_j(x(\rho)) h_j(x(\rho)) = 0.$$

Defining $\lambda_j(\rho) := -\rho \cdot h_j(x(\rho))$ one obtains

$$\nabla f(x(\rho)) - \sum_{j=1}^p \lambda_j(\rho) \nabla h_j(x(\rho)) = 0.$$

If $x(\rho)$ converges to a solution x^* that is a regular point of the constraints, then $\lambda(\rho)$ converges to the Lagrange multiplier λ^* associated with x^* .

Penalty functions suffer from the same problems of ill-conditioning as barrier functions. As the penalty parameter increases, the condition number of the Hessian matrix of $\pi(x(\rho), \rho)$ increases, tending to ∞ as $\rho \rightarrow \infty$. Therefore the unconstrained minimization problems can become increasingly difficult to solve.

8.1 PM: Example 1

Consider the problem

$$\begin{aligned} &\text{minimize} && f(x) = -x_1x_2 \\ &\text{subject to} && h(x) = x_1 + 2x_2 - 4 = 0. \end{aligned}$$

Then the penalty method solves a sequence of unconstrained minimization problems

$$\min_x \pi(x, \rho^k) = -x_1x_2 + \frac{1}{2}\rho^k(x_1 + 2x_2 - 4)^2$$

for increasing values ρ^k . The necessary conditions for optimality are

$$\begin{aligned} -x_2 + \rho(x_1 + 2x_2 - 4) &= 0 \\ -x_1 + 2\rho(x_1 + 2x_2 - 4) &= 0. \end{aligned}$$

For $\rho > 1/4$ this yields the solution

$$\begin{aligned} x_1(\rho) &= \frac{8\rho}{4\rho - 1}, \\ x_2(\rho) &= \frac{4\rho}{4\rho - 1}, \end{aligned}$$

which is a local as well as a global minimizer. (The unconstrained problem has no minimum if $\rho \leq 1/4$.) Note that $x(\rho)$ is infeasible to the original constrained problem, since

$$h(x(\rho)) = x_1 + 2x_2 - 4 = \frac{16\rho}{4\rho - 1} - 4 = \frac{4\rho}{4\rho - 1}.$$

At any solution $x(\rho)$ we can define a Lagrange multiplier estimate

$$\lambda = -\rho \cdot h(x(\rho)) = \frac{-4\rho}{4\rho - 1}.$$

As ρ tends to ∞ one obtains

$$\lim_{\rho \rightarrow \infty} x_1(\rho) = \lim_{\rho \rightarrow \infty} \frac{2}{1 - 1/4\rho} = 2, \quad \lim_{\rho \rightarrow \infty} x_2(\rho) = \lim_{\rho \rightarrow \infty} \frac{1}{1 - 1/4\rho} = 1,$$

and indeed $x^* = (2, 1)^T$ is the minimizer for the constrained problem. Further,

$$\lim_{\rho \rightarrow \infty} \lambda(\rho) = \lim_{\rho \rightarrow \infty} \frac{-1}{1 - 1/4\rho} = -1,$$

and indeed $\lambda^* = -1$ is the Lagrange multiplier at x^* .

The ill-conditioning of the penalty function is demonstrated by the Hessian matrix at $x(\rho)$:

$$\nabla_x^2 \pi(x(\rho), \rho) = \begin{pmatrix} \rho & 2\rho - 1 \\ 2\rho - 1 & 4\rho \end{pmatrix}.$$

It can be shown that its condition number is approximately $25\rho/4$. When ρ is large, the Hessian matrix is ill-conditioned.

It is also possible to apply penalty methods to problems with inequality constraints. Consider for example the problem

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g_i(x) \geq 0, \quad i = 1, \dots, m. \end{aligned}$$

For example, the **quadratic-loss penalty** in this case is

$$\psi(x) := \frac{1}{2} \sum_{i=1}^m [\min(g_i(x), 0)]^2.$$

This function has continuous first derivatives

$$\nabla \psi(x) = \sum_{i=1}^m [\min(g_i(x), 0)] \nabla g_i(x),$$

but its second derivatives can be discontinuous at points where some constraint g_i is satisfied exactly. Hence, a careful implementation is necessary. One cannot safely use Newton's method to minimize the function. For this reason, straightforward penalty methods have not been widely used for solving general inequality-constrained problems.

For a convergence discussion of penalty methods, see [GNS08] (chapter 16.2.3).

LINEAR PROGRAMMING

Linear Programming (LP; same abbreviation for “Linear Program”) is a linear optimization technique which became widely known for **resource planning tasks** after World War II. As the name suggests, the objective function and (in-)equality constraint functions are linear.

The term “programming” should be understood as “planning” or “conception” , rather than the action of writing a program in a particular programming language.

In this work the following **LP standard form** is chosen:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0, \end{array}$$

where $c, x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$.

9.1 Meeting other LP standard forms

To compute an optimal LP solution, there exist many functions or programs, which are referred to in short as **solver**. An LP solver usually specifies an own standard form. In the following a few **conversion techniques** are described.

- “minimize $c^T x$ ” is equivalent to “maximize $-c^T x$ ” . **Flip the sign** of the computed objective value.
- **Slack variables** $s \in \mathbb{R}^m$ help expressing inequality constraints as equality constraints:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \leq b \\ & x \geq 0 \end{array} \quad \Longleftrightarrow \quad \begin{array}{ll} \text{minimize} & c^T x + [0]^T s \\ \text{subject to} & Ax + Is = b \\ & x \geq 0 \\ & s \geq 0 \end{array}$$

or

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \geq b \\ & x \geq 0 \end{array} \quad \Longleftrightarrow \quad \begin{array}{ll} \text{minimize} & c^T x + [0]^T s \\ \text{subject to} & Ax - Is = b \\ & x \geq 0 \\ & s \geq 0 \end{array}$$

where I is the $m \times m$ unit matrix (main diagonal elements are 1, 0 otherwise) and $[0] \in \mathbb{R}^m$ is a vector of zeros.

- **Free variables** $y \in \mathbb{R}^k$ can be modeled as difference of two non-negative LP variables $y := y^+ - y^-$:

$$\begin{array}{ll} \text{minimize} & c^T x + d^T y \\ \text{subject to} & Ax + By = b \\ & x \geq 0 \\ & y \text{ "free"} \end{array} \quad \Longleftrightarrow \quad \begin{array}{ll} \text{minimize} & c^T x + d^T y^+ - d^T y^- \\ \text{subject to} & Ax + By^+ - By^- = b \\ & x \geq 0 \\ & y^+ \geq 0 \\ & y^- \geq 0 \end{array}$$

However, due to the introduced redundancy this approach might fail in practical applications and other techniques are known, see [LY16] (part I).

- Expressing **box constraints** $l, x, u \in \mathbb{R}^n$.

$$\begin{array}{ll}
 \text{minimize} & c^T x \\
 \text{subject to} & Ax = b \\
 & l \leq x \leq u
 \end{array}
 \quad \Longleftrightarrow \quad
 \begin{array}{ll}
 \text{minimize} & c^T x \\
 \text{subject to} & Ax = b \\
 & -Ix \leq -l \\
 & Ix \leq u \\
 & x \text{ "free"}
 \end{array}$$

where I is the $n \times n$ unit matrix (main diagonal elements are 1, 0 otherwise). See also the note about “free variables” above.

- Expressing equality constraints with inequality constraints.

$$\begin{array}{ll}
 \text{minimize} & c^T x \\
 \text{subject to} & Ax = b \\
 & x \geq 0
 \end{array}
 \quad \Longleftrightarrow \quad
 \begin{array}{ll}
 \text{minimize} & c^T x \\
 \text{subject to} & Ax \leq b \text{ (} +\varepsilon \text{)} \\
 & Ax \geq b \text{ (} -\varepsilon \text{)} \\
 & x \geq 0
 \end{array}$$

The resulting problem is usually ill-posed and a small tolerance $\varepsilon > 0$ can be added to each element of b for numerical stability.

For more information on LP and equivalent formulations, see [LY16] (part I) and [BV04] (chapter 4).

Part II

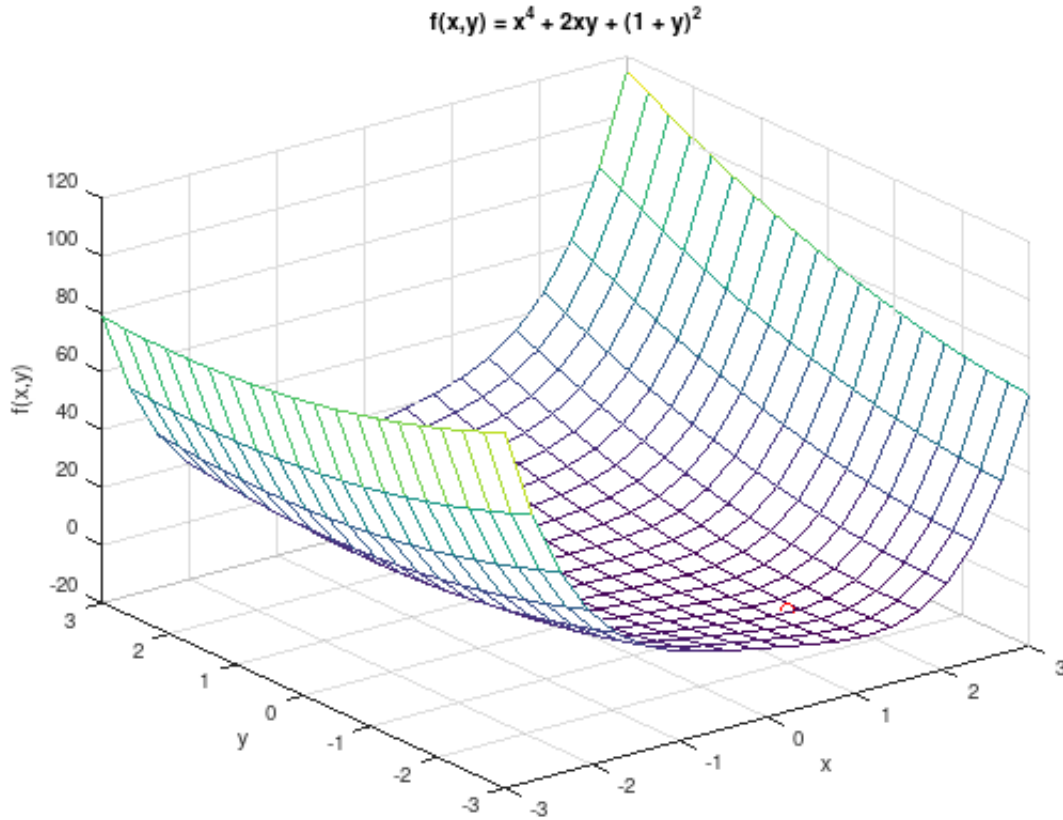
Exercises

Consider the function $f(x, y) = x^4 + 2xy + (1 + y)^2$.

1. Create a 3D plot of f for $-3 \leq x, y \leq 3$.

```
f = @(x,y) x.^4 + 2.*x.*y + (1 + y).^2;

[x, y] = meshgrid(linspace(-3, 3, 20));
mesh(x, y, f(x,y));
grid on;
xlabel('x');
ylabel('y');
zlabel('f(x,y)');
title('f(x,y) = x^4 + 2xy + (1 + y)^2');
hold on;
plot3(1, -2, -2, 'ro');
```



2. Compute the gradient ∇f and the Hessian matrix $\nabla^2 f$.

Solution:

$$\nabla f(x, y) = \begin{pmatrix} \frac{\partial}{\partial x} f(x, y) \\ \frac{\partial}{\partial y} f(x, y) \end{pmatrix} = \begin{pmatrix} 4x^3 + 2y \\ 2x + 2(1 + y) \end{pmatrix}$$

$$\nabla^2 f(x, y) = \begin{pmatrix} \frac{\partial^2}{\partial x \partial x} f(x, y) & \frac{\partial^2}{\partial x \partial y} f(x, y) \\ \frac{\partial^2}{\partial y \partial x} f(x, y) & \frac{\partial^2}{\partial y \partial y} f(x, y) \end{pmatrix} = \begin{pmatrix} 12x^2 & 2 \\ 2 & 2 \end{pmatrix}$$

3. Compute all real stationary points and the respective function values.

Solution:

Stationary points: $\nabla f = 0$.

$$\begin{aligned} 4x^3 + 2y &= 0 \\ 2x + 2y + 2 &= 0 \end{aligned}$$

From the second equation follows $y = -(x + 1)$. The resulting equation

$$2x^3 - x - 1 = 0$$

yields the real stationary point $P_1 = (1, -2)^T$ with $f(P_1) = -2$.

Use polynomial long division to find more potential real roots:

$$(2x^3 - x - 1) : (x - 1) = 2x^2 + 2x + 1$$

Finally the equation

$$x^2 + x + 0.5 = 0$$

has only complex roots.

Thus P_1 is the only real stationary point.

4. Classify all stationary points.

Solution:

Hessian matrix at P_1 :

$$\nabla^2 f(P_1) = \begin{pmatrix} 12 & 2 \\ 2 & 2 \end{pmatrix}$$

Classification with eigenvalues (roots of the characteristic polynomial):

$$0 = \det \begin{pmatrix} 12 - \lambda & 2 \\ 2 & 2 - \lambda \end{pmatrix} = (12 - \lambda)(2 - \lambda) - 4 = \lambda^2 - 14\lambda + 20$$

$$\Rightarrow \lambda_{1,2} = 7 \pm \sqrt{29} \approx 7 \pm 5.385 \dots > 0$$

The roots of the characteristic polynomial are all positive. Thus the Hessian matrix is positive definite and P_1 is a **strict local minimum**.

5. Numerical experiments.

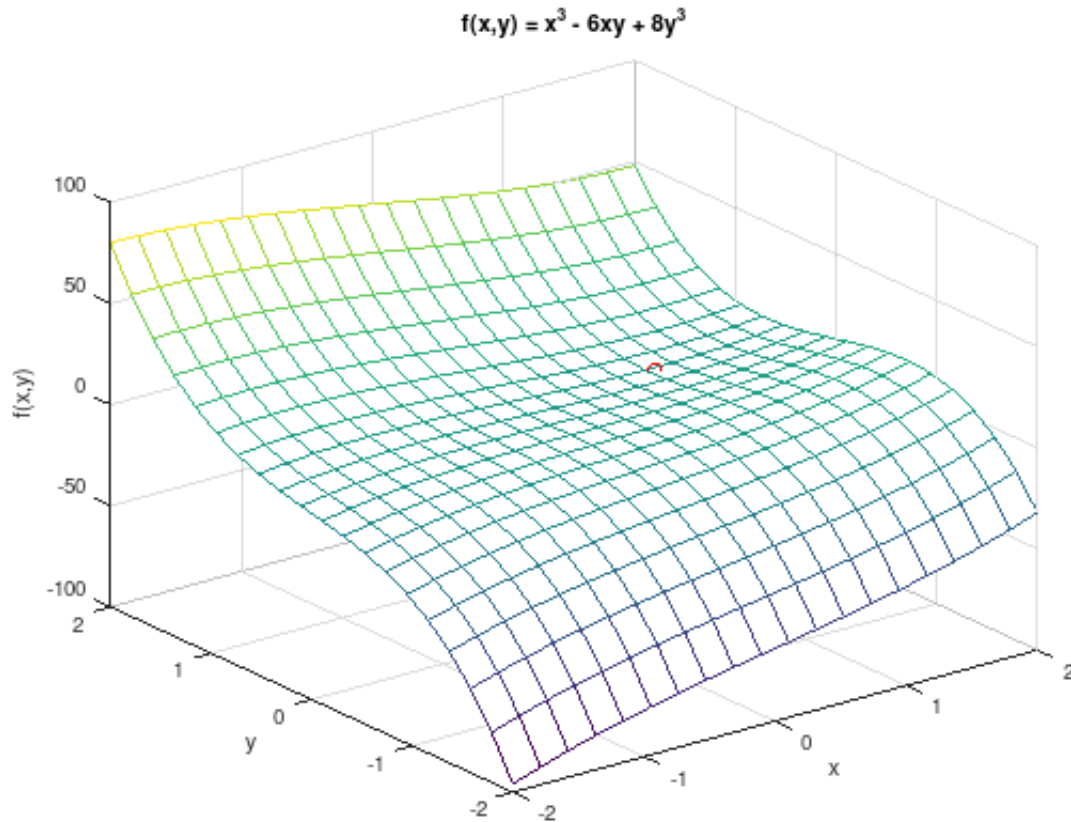
Study `um01_experiment` in Matlab/Octave.

Consider the function $f(x, y) = x^3 - 6xy + 8y^3$.

1. Create a plot of f for $-2 \leq x, y \leq 2$.

```
f = @(x,y) x.^3 - 6.*x.*y + 8.*y.^3;

[x, y] = meshgrid (linspace (-2, 2, 20));
mesh (x, y, f(x,y));
grid on;
xlabel ('x');
ylabel ('y');
zlabel ('f(x,y)');
title ('f(x,y) = x^3 - 6xy + 8y^3');
hold on;
plot3 (1, 0.5, -1, 'ro');
```



2. Compute the gradient ∇f and the Hessian matrix $\nabla^2 f$.

Solution:

$$\nabla f = \begin{pmatrix} 3x^2 - 6y \\ 24y^2 - 6x \end{pmatrix}$$

$$\nabla^2 f = \begin{pmatrix} 6x & -6 \\ -6 & 48y \end{pmatrix}$$

3. Compute all stationary points and the respective function values.

Solution:

Stationary points: $\nabla f = 0$.

$$3x^2 - 6y = 0$$

$$24y^2 - 6x = 0$$

From the first equation follows $y = \frac{1}{2}x^2$. The resulting equation

$$6x^4 - 6x = 0 \iff (x^3 - 1)x = 0$$

yields the stationary points $P_1 = (0, 0)^T$ with $f(P_1) = 0$ and $P_2 = (1, 0.5)^T$ with $f(P_2) = -1$.

4. Classify all stationary points.

Solution:

Hessian matrix at P_1 :

$$\nabla^2 f(P_1) = \begin{pmatrix} 0 & -6 \\ -6 & 0 \end{pmatrix}$$

Classification with eigenvalues (roots of the characteristic polynomial):

$$0 = \det \begin{pmatrix} -\lambda & -6 \\ -6 & -\lambda \end{pmatrix} = \lambda^2 - 36$$

$$\Rightarrow \lambda_{1/2} = \pm\sqrt{36} = \pm 6$$

The roots of the characteristic polynomial are positive and negative. Thus the Hessian matrix is indefinite and P_1 is a **saddle point**.

Hessian matrix at P_2 :

$$\nabla^2 f(P_2) = \begin{pmatrix} 6 & -6 \\ -6 & 24 \end{pmatrix}$$

Classification with eigenvalues (roots of the characteristic polynomial):

$$0 = \det \begin{pmatrix} 6-\lambda & -6 \\ -6 & 24-\lambda \end{pmatrix} = (6-\lambda)(24-\lambda) - 36 = \lambda^2 - 30\lambda + 108$$

$$\Rightarrow \lambda_{1/2} = 15 \pm 3\sqrt{13} \approx 15 \pm 10.8... > 0$$

The roots of the characteristic polynomial are all positive. Thus the Hessian matrix is positive definite and P_2 is a **strict local minimum**.

5. Numerical experiments.

Study [*um02_experiment*](#) in Matlab/Octave.

Consider the [Rosenbrock function](#)

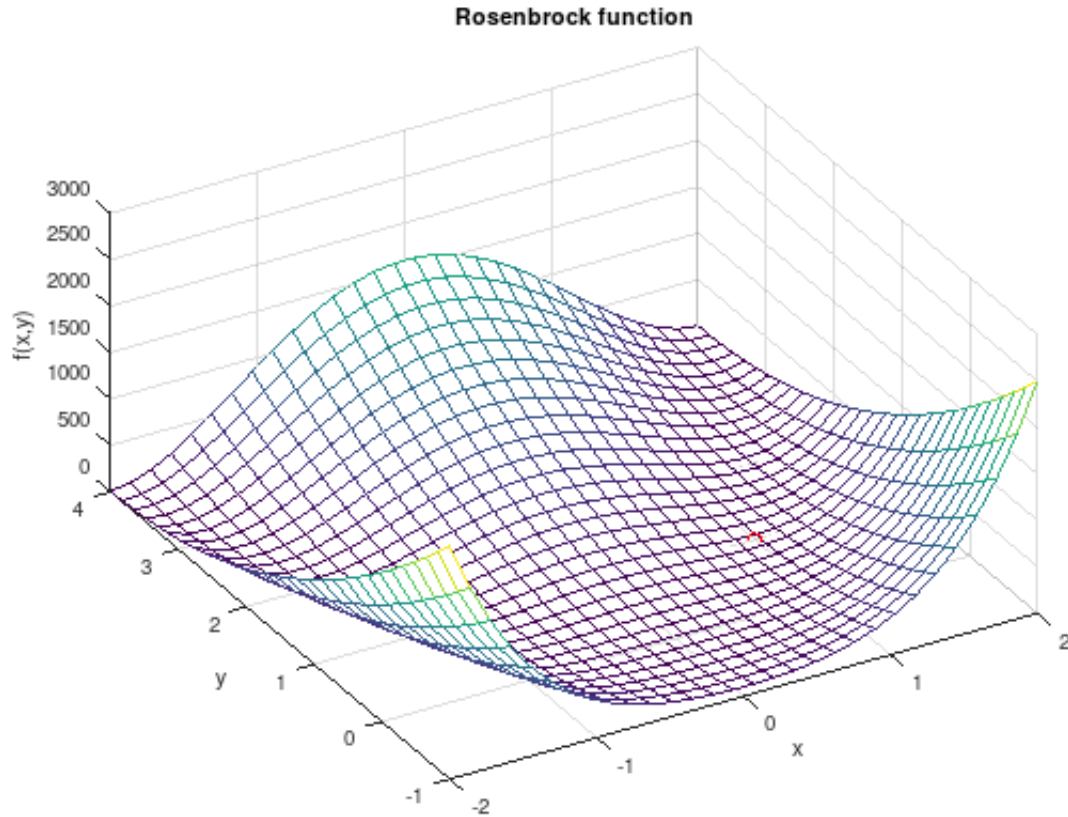
$$f(x, y) = (a - x)^2 + b(y - x^2)^2.$$

It has a global minimum at $P_1 = (a, a^2)$ with $f(P_1) = 0$. In this exercise the parameters $a = 1$ and $b = 100$ are chosen, as in many books on mathematical optimization.

$$\nabla f = \begin{pmatrix} -2(a - x) - 4bx(y - x^2) \\ 2b(y - x^2) \end{pmatrix}$$
$$\nabla^2 f = \begin{pmatrix} 2 - 4by + 12bx^2 & -4bx \\ -4bx & 2b \end{pmatrix}$$

1. Create a plot of f for $-2 \leq x \leq 2$ and $-1 \leq y \leq 4$.

```
f = @(x,y) 100 * (y - x.^2).^2 + (1 - x).^2;
[x, y] = meshgrid(linspace(-2, 2, 30), linspace(-1, 4, 30));
mesh(x, y, f(x,y));
grid on;
xlabel('x');
ylabel('y');
zlabel('f(x,y)');
title('Rosenbrock function');
hold on;
plot3(1, 1, 0, 'ro');
view(-30, 50);
```



2. Numerical experiments.

Study `um03_experiment` in Matlab/Octave. Repeat this exercise with other well-known Test functions for optimization.

Consider the following constrained optimization problem:

$$\begin{array}{llll} \text{minimize} & f(x_1, x_2) & = (x_1 - 3)^2 + (x_2 - 2)^2 & \\ \text{subject to} & g_1(x_1, x_2) & = x_1^2 + x_2^2 - 5 & \leq 0, \\ & g_2(x_1, x_2) & = x_1 + 2x_2 - 4 & \leq 0, \\ & g_3(x_1, x_2) & = -x_1 & \leq 0, \\ & g_4(x_1, x_2) & = -x_2 & \leq 0, \end{array}$$

with

$$\nabla f(x_1, x_2) = \begin{pmatrix} 2(x_1 - 3) \\ 2(x_2 - 2) \end{pmatrix}, \quad \nabla g_1(x_1, x_2) = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix}, \quad \nabla g_2(x_1, x_2) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

The optimal point is $\begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ with $f(x_1^*, x_2^*) = 2$ and $\nabla f(x_1^*, x_2^*) = \begin{pmatrix} -2 \\ -2 \end{pmatrix}$.

```
% Optimal point.
px = 2;
py = 1;

g1 = @(x) real(sqrt(5 - x.^2));
g2 = @(x) -(x - 4) ./ 2;

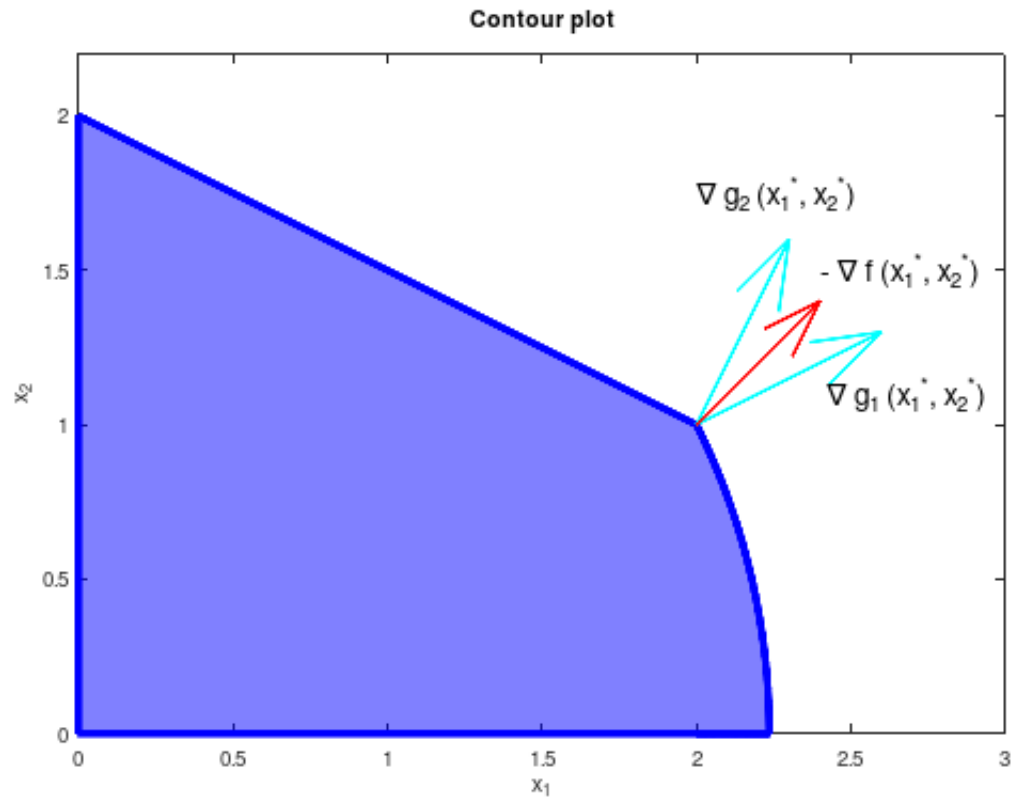
% Visualize constrained set of feasible solutions (blue).
x = linspace(2, sqrt(5), 100);
area([0 2 x], [g2(0) g2(2) g1(x)], ...
    'FaceColor', 'blue', ...
    'FaceAlpha', 0.5, ...
    'LineWidth', 4, ...
    'EdgeColor', 'blue');

% Visualize scaled gradients of objective function (red arrow)
% and constraint functions (cyan arrows).
hold on;
quiver(px, py, 0.15 * 4, 0.15 * 2, 'LineWidth', 2, 'c');
quiver(px, py, 0.3 * 1, 0.3 * 2, 'LineWidth', 2, 'c');
quiver(px, py, 0.2 * 2, 0.2 * 2, 'LineWidth', 2, 'r');
text(2.40, 1.50, '- \nabla f ({x_1}^{*}, {x_2}^{*})', 'FontSize', 14);
text(2.00, 1.75, '\nabla g_2 ({x_1}^{*}, {x_2}^{*})', 'FontSize', 14);
text(2.42, 1.10, '\nabla g_1 ({x_1}^{*}, {x_2}^{*})', 'FontSize', 14);
axis equal;
xlim([0 3.0]);
ylim([0 2.2]);
xlabel('x_1');
```

(continues on next page)

(continued from previous page)

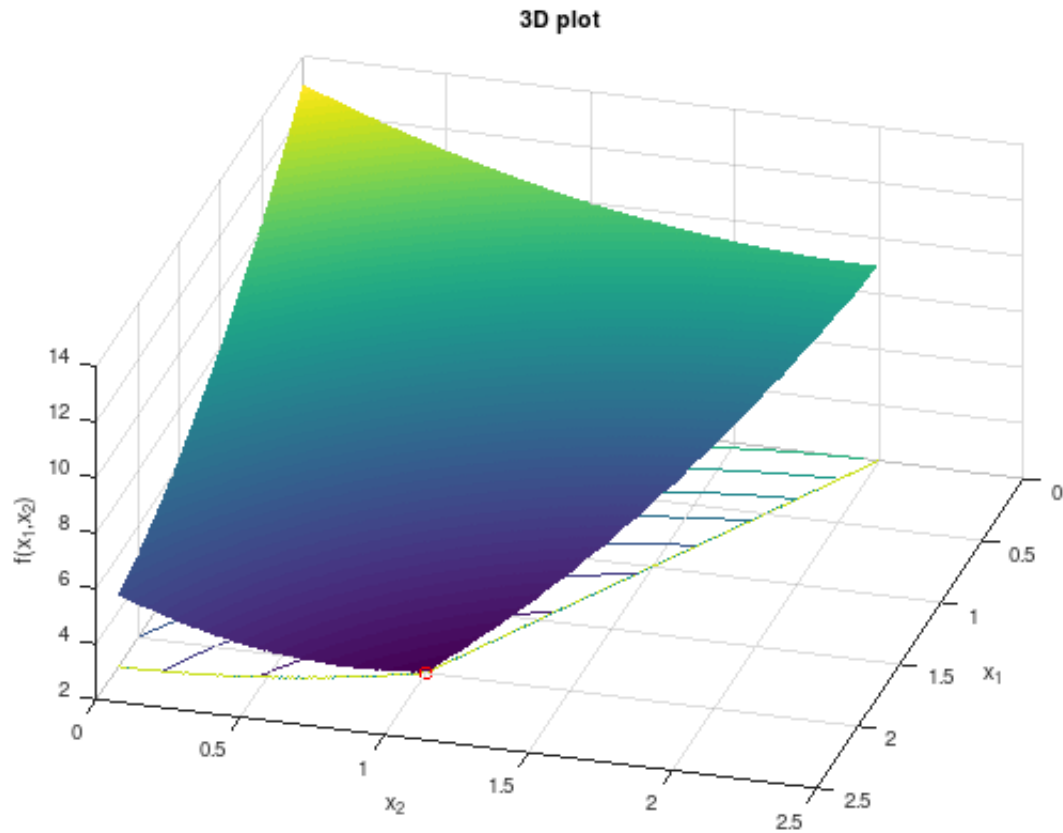
```
ylabel ('x_2');
title ('Contour plot');
```



```
% Optimal point.
px = 2;
py = 1;
pz = 2;

[X1, X2] = meshgrid (linspace (0, 2.5, 500));
FX = (X1 - 3).^2 + (X2 - 2).^2;

% Remove infeasible points.
FX((X1.^2 + X2.^2) > 5) = inf;
FX(X1 + 2*X2 > 4) = inf;
surfc (X1, X2, FX);
shading flat;
hold on;
plot3 (px, py, pz, 'ro');
xlabel ('x_1');
ylabel ('x_2');
zlabel ('f(x_1,x_2)');
title ('3D plot');
view (106, 44);
```



At the optimal point only the constraints g_1 and g_2 are active, thus $\lambda_3^* = \lambda_4^* = 0$.

According to KKT, there exist unique $\lambda_1^* \geq 0, \lambda_2^* \geq 0$ with

$$\begin{pmatrix} -2 \\ -2 \end{pmatrix} + \lambda_1^* \begin{pmatrix} 4 \\ 2 \end{pmatrix} + \lambda_2^* \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 0$$

thus $\lambda_1^* = \frac{1}{3}$ and $\lambda_2^* = \frac{2}{3}$.

13.1 Numerical experiment (only Matlab)

```
function RM01()
% Nonlinear objective function.
fun = @(x) (x(1) - 3).^2 + (x(2) - 2).^2;

% Starting point.
x0 = [2, 1];

% Linear inequality constraints A * x <= b.
A = [1 2]; % g_2
b = [4];

% Linear equality constraints Aeq * x = beq.
Aeq = [];
```

(continues on next page)

(continued from previous page)

```
beq = [];  
  
% Bounds lb <= x <= ub  
lb = [0, 0];      % g_3 and g_4  
ub = [];  
  
% Call solver.  
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon (fun,x0,A,b,Aeq,beq,lb,ub,  
↳@nonlcon);  
  
% Display interesting details.  
  
exitflag % == 1 success  
x        % optimal solution  
fval      % function value at optimal solution  
grad      % gradient of fun at optimal solution  
hessian   % Hessian matrix of fun at optimal solution  
lambda    % Lagrange parameter  
lambda.lower % lambda_3 and lambda_4  
lambda.ineqlin % lambda_2  
lambda.ineqnonlin % lambda_1  
end  
  
% Nonlinear constraint function for g_1.  
function [c,ceq] = nonlcon(x)  
    c = x(1).^2 + x(2).^2 - 5;  
    ceq = 0;  
end
```


Consider the following constrained optimization problem:

$$\begin{array}{llll}
 \text{minimize} & f(x_1, x_2) & = (x_1 - 1.25)^2 + (x_2 - 1)^2 & \\
 \text{subject to} & g_1(x_1, x_2) & = x_1^2 + x_2^2 - 1 & \leq 0, \\
 & g_2(x_1, x_2) & = x_2 - 0.25 & \leq 0, \\
 & g_3(x_1, x_2) & = -x_1 & \leq 0, \\
 & g_4(x_1, x_2) & = -x_2 & \leq 0,
 \end{array}$$

with

$$\nabla f(x_1, x_2) = \begin{pmatrix} 2(x_1 - 1.25) \\ 2(x_2 - 1) \end{pmatrix}, \quad \nabla g_1(x_1, x_2) = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix}, \quad \nabla g_2(x_1, x_2) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

The optimal point is $\begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} = \frac{1}{4} \begin{pmatrix} \sqrt{15} \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.97 \\ 0.25 \end{pmatrix}$ with $f(x_1^*, x_2^*) \approx 0.64$ and $\nabla f(x_1^*, x_2^*) = \frac{1}{2} \begin{pmatrix} \sqrt{15} - 5 \\ -3 \end{pmatrix} \approx \begin{pmatrix} -0.56 \\ -1.5 \end{pmatrix}$

```

% Optimal point.
px = 0.97;
py = 0.25;

% Visualize constrained set of feasible solutions (blue).
area ([0 1 0.97 0 0], [0 0 1/4 1/4 0], ...
      'FaceColor', 'blue', ...
      'FaceAlpha', 0.5, ...
      'LineWidth', 4, ...
      'EdgeColor', 'blue');

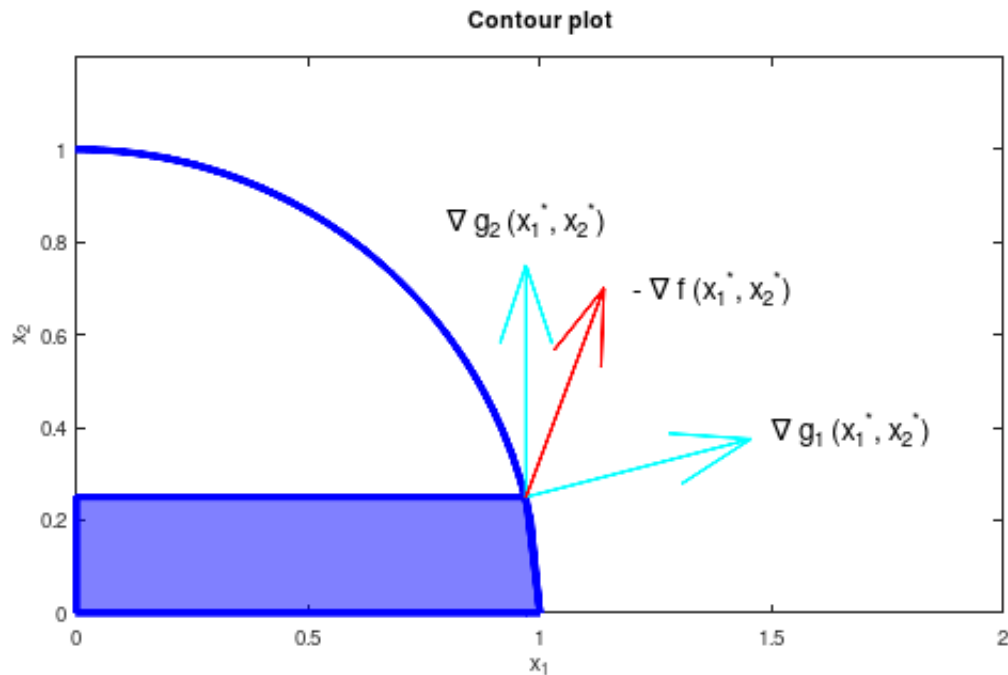
% Visualize scaled gradients of objective function (red arrow)
% and constraint functions (cyan arrows).
hold on;
x = 0:0.02:1;
plot (x, sqrt (1 - x.^2), 'LineWidth', 4, 'b');
quiver (px, py, 0.5 * px, 0.5 * py, 'LineWidth', 2, 'c');
quiver (px, py, 0, 0.5, 'LineWidth', 2, 'c');
quiver (px, py, 0.3 * 0.56, 0.3 * 1.5, 'LineWidth', 2, 'r');
text (1.2, 0.70, '\nabla f ({x_1}^{\ast}, {x_2}^{\ast})', 'FontSize', 14);
text (0.8, 0.85, '\nabla g_2 ({x_1}^{\ast}, {x_2}^{\ast})', 'FontSize', 14);
text (1.5, 0.40, '\nabla g_1 ({x_1}^{\ast}, {x_2}^{\ast})', 'FontSize', 14);
axis equal;
xlim ([0 2.0]);
ylim ([0 1.2]);

```

(continues on next page)

(continued from previous page)

```
xlabel ('x_1');
ylabel ('x_2');
title ('Contour plot');
```



```
% Optimal point.
px = 0.97;
py = 0.25;

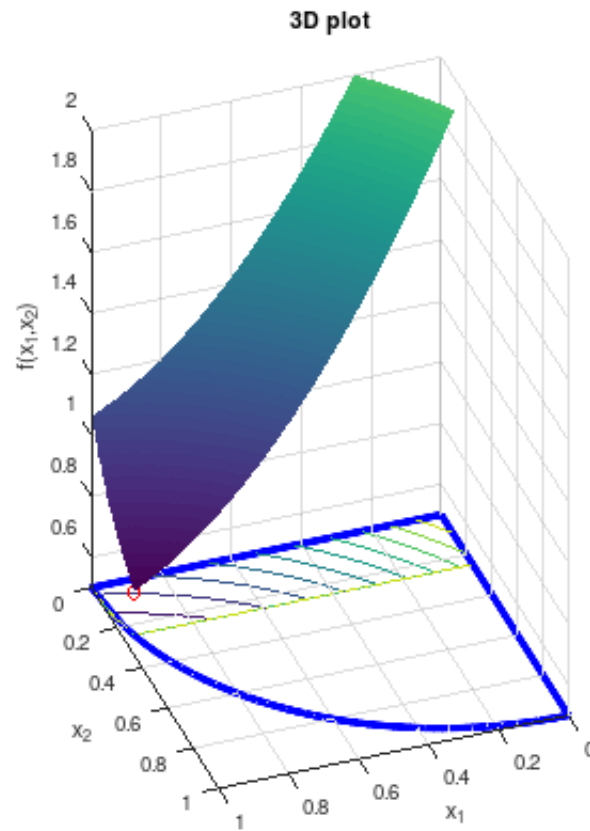
[X1, X2] = meshgrid (linspace (0, 1, 500));
FX = (X1 - 1.25).^2 + (X2 - 1).^2;

% Remove infeasible points.
FX((X1.^2 + X2.^2) > 1) = inf;
FX(X2 > 0.25) = inf;
surfc (X1, X2, FX);
shading flat;
hold on;
x = 0:0.02:1;
plot3 (x, sqrt (1 - x.^2), 0.5 .* ones (size (x)), 'LineWidth', 4, 'b');
plot3 ([1 0 0], [0 0 1], [0.5 0.5 0.5], 'LineWidth', 4, 'b');
plot3 (px, py, (px - 1.25)^2 + (py - 1)^2, 'ro');
xlabel ('x_1');
ylabel ('x_2');
zlabel ('f(x_1,x_2)');
```

(continues on next page)

(continued from previous page)

```
title ('3D plot');
axis equal;
zlim ([0.5, 2]);
view (160, 35);
```



At the optimal point only the constraints g_1 and g_2 are active, thus $\lambda_3^* = \lambda_4^* = 0$.

According to KKT, there exist unique $\lambda_1^* \geq 0$, $\lambda_2^* \geq 0$ with

$$\frac{1}{2} \begin{pmatrix} \sqrt{15} - 5 \\ -3 \end{pmatrix} + \lambda_1^* \frac{1}{2} \begin{pmatrix} \sqrt{15} \\ 1 \end{pmatrix} + \lambda_2^* \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0$$

thus $\lambda_1^* = \frac{1}{3}(\sqrt{15} - 3) \approx 0.29$ and $\lambda_2^* = \frac{12 - \sqrt{15}}{6} \approx 1.35$.

14.1 Numerical experiment (only Matlab)

```
function RM02 ()
% Nonlinear objective function.
fun = @(x) (x(1) - 1.25).^2 + (x(2) - 1).^2;

% Starting point.
x0 = [1, 0.25];
```

(continues on next page)

(continued from previous page)

```
% Linear inequality constraints A * x <= b.
A = [];
b = [];

% Linear equality constraints Aeq * x = beq.
Aeq = [];
beq = [];

% Bounds lb <= x <= ub
lb = [0, 0];      % g_3 and g_4
ub = [10, 0.25]; % g_2

% Call solver.
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon (fun,x0,A,b,Aeq,beq,lb,ub,
↳@nonlcon);

% Display interesting details.

exitflag % == 1 success
x        % optimal solution
fval     % function value at optimal solution
grad     % gradient of fun at optimal solution
hessian  % Hessian matrix of fun at optimal solution
lambda   % Lagrange parameter
lambda.lower % lambda_3 and lambda_4
lambda.upper(2) % lambda_2
lambda.ineqnonlin % lambda_1
end

% Nonlinear constraint function for g_1.
function [c,ceq] = nonlcon(x)
    c = x(1).^2 + x(2).^2 - 1;
    ceq = 0;
end
```

Consider the following constrained optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(x_1, x_2) = x_1 + x_2 \\ \text{subject to} \quad & g_1(x_1, x_2) = (x_1 - 1)^2 + x_2^2 - 1 = 0, \\ & g_2(x_1, x_2) = (x_1 - 2)^2 + x_2^2 - 4 = 0, \end{aligned}$$

with

$$\nabla f(x_1, x_2) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \nabla g_1(x_1, x_2) = \nabla g_2(x_1, x_2) = \begin{pmatrix} 2(x_1 - 1) \\ 2x_2 \end{pmatrix}.$$

The optimal point is $\begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ with $f(x_1^*, x_2^*) = 0$.

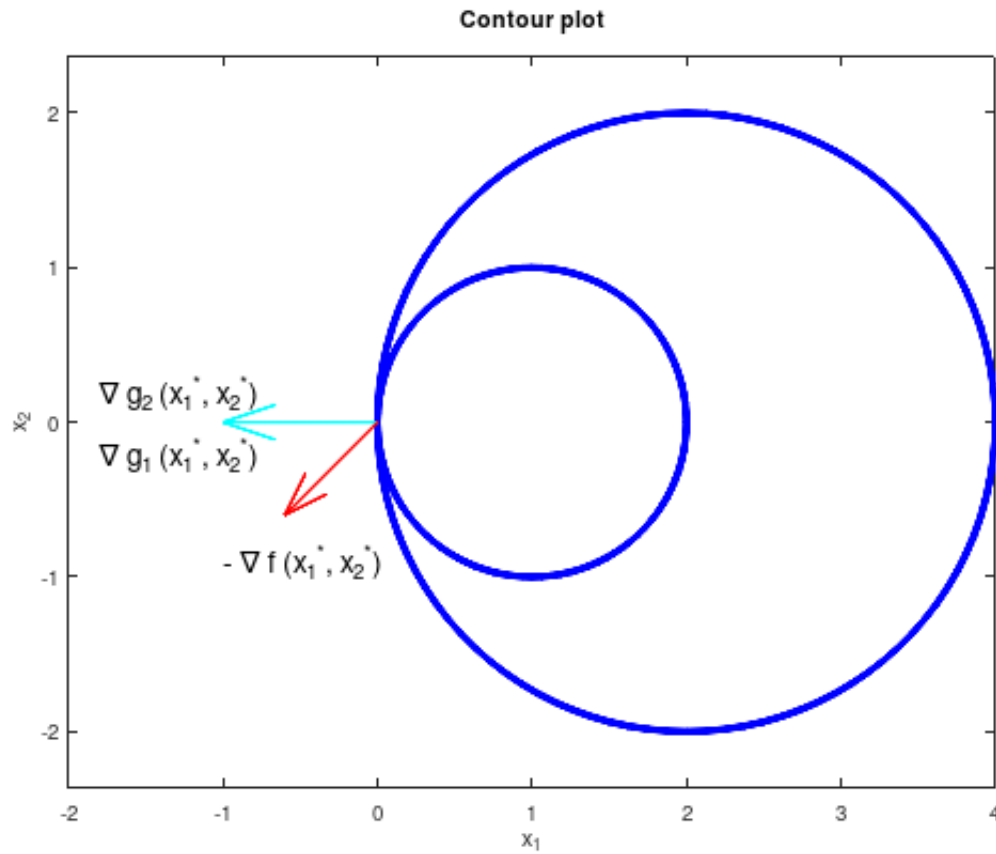
However, because $\nabla g_1(x_1, x_2) = \nabla g_2(x_1, x_2)$ are not linear independent, the regularity constraint qualification (LICQ) is violated. The KKT conditions cannot be satisfied.

```
% Optimal point.
px = 0;
py = 0;

function circle (x, y, r)
    t = 0:(pi / 50):2*pi;
    x = r * cos (t) + x;
    y = r * sin (t) + y;
    plot (x, y, 'b', 'LineWidth', 4);
end

% Visualize constrained set of feasible solutions (blue).
circle (1, 0, 1);
hold on;
circle (2, 0, 2);

% Visualize scaled gradients of objective function (red arrow)
% and constraint functions (cyan arrows).
quiver (px, py, -1, 0, 'LineWidth', 2, 'c');
quiver (px, py, -0.6, -0.6, 'LineWidth', 2, 'r');
text (-1.0, -0.9, '- \nabla f ({x_1}^{*}, {x_2}^{*})', 'FontSize', 14);
text (-1.8, 0.2, '\nabla g_2 ({x_1}^{*}, {x_2}^{*})', 'FontSize', 14);
text (-1.8, -0.2, '\nabla g_1 ({x_1}^{*}, {x_2}^{*})', 'FontSize', 14);
axis equal;
xlim ([-2 4]);
xlabel ('x_1');
ylabel ('x_2');
title ('Contour plot');
```



15.1 Numerical experiment (only Matlab)

```
function RM03()
    % Nonlinear objective function.
    fun = @(x) x(1) + x(2);

    % Starting point.
    x0 = [0, 0];

    % Linear inequality constraints A * x <= b.
    A = [];
    b = [];

    % Linear equality constraints Aeq * x = beq.
    Aeq = [];
    beq = [];

    % Bounds lb <= x <= ub
    lb = [];
    ub = [];

    % Call solver.
    [x,fval,exitflag,output,lambda,grad,hessian] = fmincon (fun,x0,A,b,Aeq,beq,lb,ub,
    @nonlcon);
```

(continues on next page)

(continued from previous page)

```

% Display interesting details.

exitflag % == 1 success
output
x        % optimal solution
fval     % function value at optimal solution
grad     % gradient of fun at optimal solution
hessian  % Hessian matrix of fun at optimal solution
lambda   % Lagrange parameter
lambda.eqnonlin % lambda_1 and lambda_2

disp ('"-Constraints')
[(x(1) - 1).^2 + x(2).^2 - 1; ...
 (x(1) - 2).^2 + x(2).^2 - 4]
end

% Nonlinear constraint function for g_1.
function [c,ceq] = nonlcon(x)
    c = 0;
    ceq = [(x(1) - 1).^2 + x(2).^2 - 1; ...
           (x(1) - 2).^2 + x(2).^2 - 4];
end

```


Consider the following constrained optimization problem:

$$\begin{array}{ll} \text{minimize} & f(x_1, x_2) := e^{3x_1} + e^{-4x_2} \\ \text{subject to} & h(x_1, x_2) := x_1^2 + x_2^2 - 1 = 0. \end{array}$$

The corresponding Lagrangian is:

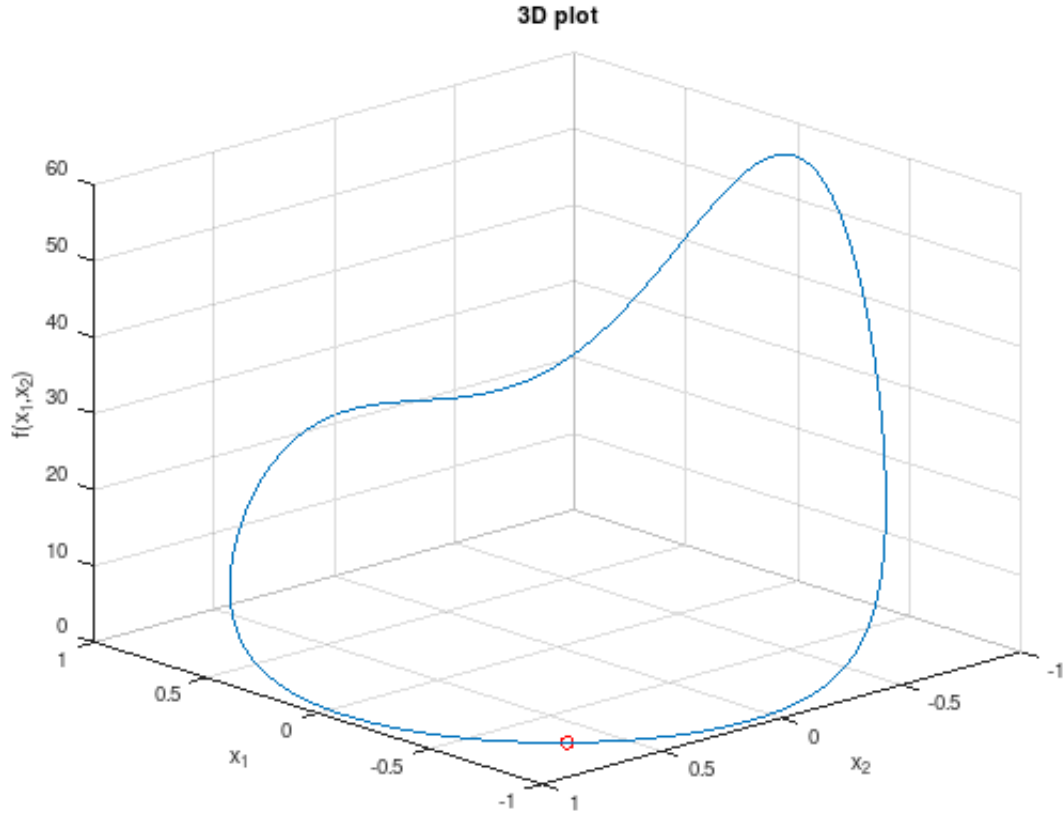
$$L(x_1, x_2, \mu) = e^{3x_1} + e^{-4x_2} + \mu(x_1^2 + x_2^2 - 1)$$

and the KKT optimality conditions:

$$\begin{aligned} \nabla_{x_1} L(x_1, x_2, \mu) &:= 3e^{3x_1} + 2\mu x_1 = 0, \\ \nabla_{x_2} L(x_1, x_2, \mu) &:= -4e^{-4x_2} + 2\mu x_2 = 0, \\ h(x) = \nabla_{\mu} L(x_1, x_2, \mu) &:= x_1^2 + x_2^2 - 1 = 0. \end{aligned}$$

```
% Optimal point.
px = -0.75;
py = 0.66;

theta = 0:0.02:2*pi;
x = cos (theta);
y = sin (theta);
z = exp (3*x) + exp (-4*y);
plot3 (x,y,z);
hold on;
plot3 (px, py, exp (3*px) + exp (-4*py), 'ro');
xlabel ('x_1');
ylabel ('x_2');
zlabel ('f(x_1,x_2)');
title ('3D plot');
grid on;
view (-133, 23);
```



16.1 Sequential Quadratic Programming (SQP)

Formulate the KKT optimality conditions of the quadratic sub-problem (6.1) in matrix form, which correspond to the following non-linear system of equations:

$$\begin{pmatrix} Q & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta \mu \end{pmatrix} = - \begin{pmatrix} \nabla_x L(x, \mu) \\ h(x) \end{pmatrix},$$

with $B = \nabla^T h(x) = (2x_1, 2x_2)$ and $Q = \nabla_{x,x}^2 L(x, \mu) = \begin{pmatrix} 9e^{3x_1} + 2\mu & 0 \\ 0 & 16e^{-4x_2} + 2\mu \end{pmatrix}$.

For the starting point $\mathbf{x}_0 = (x_0, y_0, \mu_0)^T = (-1, 1, 1)^T$ the linear system to solve to compute the first Newton correction is:

$$\begin{pmatrix} 2.44808 & 0 & -2 \\ 0 & 2.29305 & 2 \\ -2 & 2 & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \mu \end{pmatrix} = \begin{pmatrix} 1.8506 \\ -1.9267 \\ -1 \end{pmatrix}$$

16.2 Numerical experiment

```

format shortE

f = @(x) exp(3*x(1)) + exp(-4*x(2));
h = @(x) x(1)^2 + x(2)^2 - 1;

% Lagrange multiplier is x(3).

grad_L = @(x) [ 3*exp( 3*x(1)) + 2*x(3)*x(1);
               -4*exp(-4*x(2)) + 2*x(3)*x(2)];

B = @(x) 2 * [x(1), x(2)];

Q = @(x) [9*exp(3*x(1)) + 2*x(3), 0;
          0, 16*exp(-4*x(2)) + 2*x(3)];

% Initial values.
x0 = [-1, 1, 1]';
disp('          x_k          mu_k          ||grad_x L||          ||h||')
disp([x0', norm(grad_L(x0)), norm(h(x0))])

% Newton's method, SQP iteration.
x = x0;
for i = 1:5
    A = [ Q(x), B(x)'; ...
          B(x),    0    ];
    b = [-grad_L(x); -h(x)];
    x = x + A \ b;
    disp([x', norm(grad_L(x)), norm(h(x))])
end







```

	x_k	mu_k	grad_x L	h
-1.0000e+00	1.0000e+00	1.0000e+00	2.6716e+00	1.0000e+00
-7.7423e-01	7.2577e-01	3.5104e-01	3.8268e-01	1.2617e-01
-7.4865e-01	6.6614e-01	2.1606e-01	1.1101e-02	4.2107e-03
-7.4834e-01	6.6332e-01	2.1232e-01	3.9448e-06	8.0212e-06
-7.4834e-01	6.6332e-01	2.1232e-01	3.1054e-11	1.6306e-11
-7.4834e-01	6.6332e-01	2.1232e-01	1.2413e-16	0

17.1 Collection of Linear Programming (LP) exercises

See the *section about Linear Programming* before working on the exercises.

17.2 Example 1: Car factory

				
Variables		Assembly time	Painting time	Profit margin per car
	Cheapo (x)	30 minutes	30 minutes	\$10,000
	Deluxe (y)	100 minutes	30 minutes	\$20,000
	Budget	20,000 minutes	8,000 minutes	

Source: quantamagazine (2021-11-01).

```
c = -[ 10000 20000 ]; % Maximize profit margin
A = [ 30 100; ...
      30 30 ];
```

(continues on next page)

(continued from previous page)

```
b = [ 20000; 8000 ];
Aeq = []; % No equality constraints
beq = [];
lb = [ 0 0 ];
ub = [ inf inf ];
CTYPE = repmat('U', 2, 1); % Octave: A(i,:)*x <= b(i)
x0 = []; % default start value
[x,fval,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub,x0) % Matlab: exitflag=1 success
[x,fval,exitflag] = glpk(c,A,b,lb,ub,CTYPE) % Octave: exitflag=0 success
```

```
x =

    95.238
   171.429

fval = -4.3810e+06
exitflag = 0
```

This result is of course very “technical” and needs interpretation to be useful for the car company.

The quantity of produced cars should be integers. A safe strategy is rounding the results down:

```
cheapo = floor(x(1))
deluxe = floor(x(2))
```

```
cheapo = 95
deluxe = 171
```

The computed profit `fval` is negative. However to adapt to the standard form of a minimization problem, the objective function was negated.

Furthermore, after rounding the values, the new profit can be computed from the objective function:

```
profit = -c * [cheapo; deluxe] % dollar
```

```
profit = 4370000
```

Finally, the budgets for the painting and assembly line time can be checked with `A` and `b`:

```
required_time = A * [cheapo; deluxe] % minutes
```

```
required_time =

    19950
     7980
```

```
permitted_time = b % minutes
```

```
permitted_time =

    20000
     8000
```

Thus the problem is optimally solved. Not a single deluxe or cheapo could be manufactured with the given time budget:

```
remaining_budget = permitted_time - required_time % minutes
```

```
remaining_budget =  
50  
20
```

17.3 Example 2: Alloy production

Source: [LY16] (p. 27).

A manufacturer wishes to produce an alloy that is, by weight, 30 % metal A and 70 % metal B. Five alloys are available at various prices as indicated below:

Alloy	1	2	3	4	5
%A	10	25	50	75	95
%B	90	75	50	25	5
Price [\$]/kg	5	4	3	2	1.5

The desired alloy will be produced by combining some of the other alloys. The manufacturer wishes to find the amounts of the various alloys needed and to determine the least expensive combination. Formulate this problem as a linear program.

```
c = [ 5 4 3 2 1.5 ]; % Minimize price  
A = []; % No inequality constraints  
b = [];  
Aeq = [ 10 25 50 75 95; ...  
        90 75 50 25 5 ];  
beq = [ 30; 70 ];  
lb = zeros(5,1);  
ub = +inf(5,1);  
CTYPE = repmat('S', 2, 1); % Octave: A(i,:)*x = b(i)  
x0 = []; % default start value  
%[x,fval,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub,x0) % Matlab: exitflag=1 success  
[x,fval,exitflag] = glpk(c,Aeq,beq,lb,ub,CTYPE) % Octave: exitflag=0 success
```

```
x =  
  
0  
0.9000  
0  
0.1000  
0  
  
fval = 3.8000  
exitflag = 0
```

17.4 Example 3: Oil refinery

Source: [LY16] (p. 28).

An oil refinery has two sources of crude oil: a light crude that costs 35 dollar/barrel and a heavy crude that costs 30 dollar/barrel. The refinery produces gasoline, heating oil, and jet fuel from crude in the amounts per barrel indicated in the following table:

	Gasoline	Heating oil	Jet fuel
Light crude	0.3	0.4	0.3
Heavy crude	0.3	0.2	0.2

The refinery has contracted to supply 900,000 barrels of gasoline, 800,000 barrels of heating oil, and 500,000 barrels of jet fuel. The refinery wishes to find the amounts of light and heavy crude to purchase so as to be able to meet its obligations at minimum cost. Formulate this problem as a linear program.

```
% Formulation with equality constraints not solvable.
```

```
c = [ 35 30 ]; % Minimize price
A = []; % No inequality constraints
b = [];
Aeq = [ 0.3 0.3; ...
        0.4 0.2; ...
        0.3 0.2 ];
beq = [ 900000; 800000; 500000 ];
lb = zeros(2,1);
ub = +inf(2,1);
CTYPE = repmat('S', 3, 1); % Octave: A(i,:)*x = b(i)
x0 = []; % default start value
[x,fval,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub,x0) % Matlab: exitflag=1 success
[x,fval,exitflag] = glpk(c,Aeq,beq,lb,ub,CTYPE) % Octave: exitflag=0 success
```

```
glp_simplex: unable to recover undefined or non-optimal solution
x =

    NA
    NA

fval = NA
exitflag = 10
```

```
% Formulation with inequality constraints.
```

```
c = [ 35 30 ]; % Minimize price
A = -[ 0.3 0.3; ...
        0.4 0.2; ...
        0.3 0.2 ];
b = -[ 900000; 800000; 500000 ];
Aeq = []; % No equality constraints
beq = [];
lb = zeros(2,1);
ub = +inf(2,1);
CTYPE = repmat('U', 3, 1); % Octave: -A(i,:)*x <= -b(i)
x0 = []; % default start value
[x,fval,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub,x0) % Matlab: exitflag=1 success
[x,fval,exitflag] = glpk(c,A,b,lb,ub,CTYPE) % Octave: exitflag=0 success
```



```
x =

    1.0000e+06
    2.0000e+06

fval = 9.5000e+07
exitflag = 0
```

```
-A * x + b % barrel overproduction
```

```
ans =

     0
     0
200000
```

17.5 Example 4: Car parts

Source: [LY16] (p. 28-29).

A small firm specializes in making five types of spare automobile parts. Each part is first cast from iron in the casting shop and then sent to the finishing shop where holes are drilled, surfaces are turned, and edges are ground. The required worker-hours (per 100 units) for each of the parts of the two shops are shown below:

Part	1	2	3	4	5
Casting	2	1	3	3	1
Finishing	3	2	2	1	1

The profits (in dollar per 100 units) from the parts are 30, 20, 40, 25, and 10, respectively. The capacities of the casting and finishing shops over the next month are 700 and 1,000 worker-hours, respectively. Formulate the problem of determining the quantities of each spare part to be made during the month so as to maximize profit.

```
c = -[ 30 20 40 25 10 ]; % Maximize profit
A = []; % No inequality constraints
b = [];
Aeq = [ 2 1 3 3 1; ...
        3 2 2 1 1 ];
beq = [ 700; 1000 ];
lb = zeros(5,1);
ub = +inf(5,1);
CTYPE = repmat('S', 2, 1); % Octave: A(i,:)*x = b(i)
x0 = []; % default start value
%[x,fval,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub,x0) % Matlab: exitflag=1 success
[x,fval,exitflag] = glpk(c,Aeq,beq,lb,ub,CTYPE) % Octave: exitflag=0 success
```

```
x =

     0
    400
    100
     0
     0
```

(continues on next page)

(continued from previous page)

```
fval = -12000
exitflag = 0
```

17.6 Example 5: Transport cost minimization

Source: [LY16] (p. 29).

A large textile firm has two manufacturing plants, two sources of raw material, and three market centers. The transportation costs (dollar per ton) between the sources and the plants and between the plants and the markets are as follows:

Plant	A	B	
Source 1	1	1.5	
Source 2	2	1.5	
Market	1	2	3
Plant A	4	2	1
Plant B	3	4	2

Ten tons are available from source 1 and 15 tons from source 2. The three market centers require 8 tons, 14 tons, and 3 tons. The plants have unlimited processing capacity.

- Formulate the problem of finding the shipping patterns from sources to plants to markets that minimizes the total transportation cost.
- Reduce the problem to a single standard transportation problem with two sources and three destinations. (Hint: Find minimum cost paths from sources to markets.)

```
A1 = [ 1 1.5; ...
       2 1.5 ];
A2 = [ 4 2 1; ...
       3 4 2 ];
% Minimize transport cost.
%
% x = [ S1 to A to M1,
%       to M2,
%       to M3,
%       S1 to B to M1,
%       to M2,
%       to M3,
%       S2 to A to M1,
%       to M2,
%       to M3,
%       S2 to B to M1,
%       to M2,
%       to M3 ]
c = [ A1(1,1) + A2(1,:), ...
      A1(1,2) + A2(2,:), ...
      A1(2,1) + A2(1,:), ...
      A1(2,2) + A2(2,:) ];
A = []; % No inequality constraints
b = [];
Aeq = [ 1 1 1 1 1 1 0 0 0 0 0 0; ... % Sum of Source 1
        0 0 0 0 0 0 1 1 1 1 1 1; ... % Sum of Source 2
```

(continues on next page)

(continued from previous page)

```

        1 0 0 1 0 0 1 0 0 1 0 0; ... % Sum of Market 1
        0 1 0 0 1 0 0 1 0 0 1 0; ... % Sum of Market 2
        0 0 1 0 0 1 0 0 1 0 0 1 ]; % Sum of Market 3
beq = [ 10; 15; 8; 14; 3 ];
lb = zeros(12,1);
ub = +inf(12,1);
CTYPE = repmat ('S', 5, 1); % Octave: A(i,:)*x = b(i)
x0 = []; % default start value
%[x,fval,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub,x0); % Matlab: exitflag=1 success
[x,fval,exitflag] = glpk(c,Aeq,beq,lb,ub,CTYPE); % Octave: exitflag=0 success
x'
fval
exitflag

```

```

ans =

    0    10     0     0     0     0     0     4     3     8     0     0

fval = 91
exitflag = 0

```


18.1 Linear classification

See also [BV04] (chapter 8.6).

18.1.1 Creation of labeled measurements (“training data”)

Create m random data pairs (w_i, l_i) of weights and lengths.

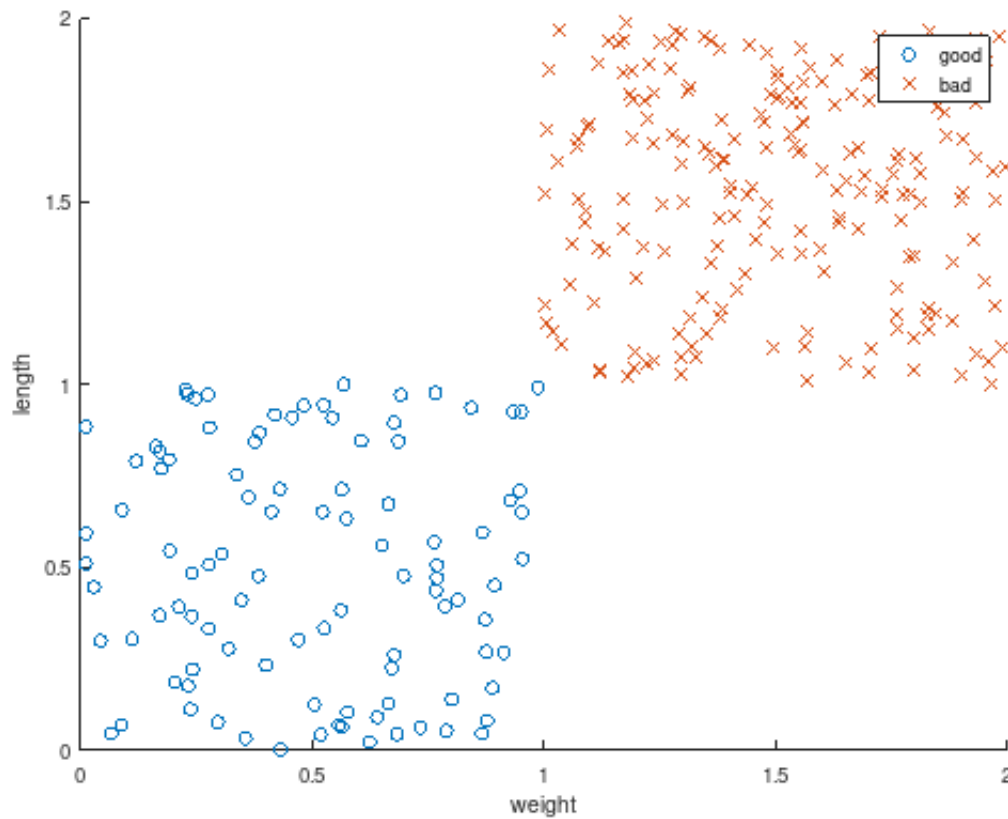
```
m = 300;  
w = rand(m,1); % weight  
l = rand(m,1); % length
```

Split the measurements into two sets, such that a data pair (w_i, l_i) is considered as “good” for $1 \leq i \leq k \leq m$ and “bad” otherwise.

```
k = m / 3;  
w = [w(1:k); w((k+1):m) + 1];  
l = [l(1:k); l((k+1):m) + 1];
```

Plot the resulting data to discriminate.

```
axis equal  
scatter(w(1:k), l(1:k), 'o'); % good  
hold on;  
scatter(w(k+1:m), l(k+1:m), 'x'); % bad  
xlabel('weight');  
ylabel('length');  
legend({'good', 'bad'});
```



18.1.2 Classification

Looking at the figure above, it seems possible to find a separating hyperplane. That is a linear function with unknown coefficients x_0 , x_1 , and x_2

$$f_{\text{linear}}(w_i, l_i) = x_0 + \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} w_i \\ l_i \end{pmatrix} = x_0 + x_1 \cdot w_i + x_2 \cdot l_i$$

to separate

- the “**good**” blue circles: $f_{\text{linear}}(w_i, l_i) \leq -1$ for $1 \leq i \leq k \leq m$ and
- the “**bad**” red crosses: $f_{\text{linear}}(w_i, l_i) \geq 1$ for $k < i \leq m$.

Note: w_i and l_i are the variables of $f_{\text{linear}}(w_i, l_i)$!

18.1.3 Feasibility problem as Linear Program (LP)

This classification problem can be formulated as LP:

$$\begin{array}{ll} \text{minimize} & \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} x \\ \text{subject to} & \begin{pmatrix} 1 & w_i & l_i \end{pmatrix} x \leq -1, \quad i = 1, \dots, k, \\ & -\begin{pmatrix} 1 & w_i & l_i \end{pmatrix} x \leq -1, \quad i = k+1, \dots, m, \end{array}$$

```
c = [0 0 0];
A = [ ones(k, 1), w(1:k), l(1:k); ...
      -ones(m-k, 1), -w(k+1:m), -l(k+1:m) ];
b = -ones(m, 1);
Aeq = []; % No equality constraints
beq = [];
lb = -inf(3, 1); % x0 to x2 are free variables
ub = inf(3, 1);
CTYPE = repmat('U', m, 1); % Octave: A(i,:)*x <= b(i)
x0 = []; % default start value
[x,~,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub,x0); % Matlab: exitflag=1 success
[x,~,exitflag] = glpk(c,A,b,lb,ub,CTYPE) % Octave: exitflag=0 success
```

```
x =

    -23.712
     10.969
     11.999

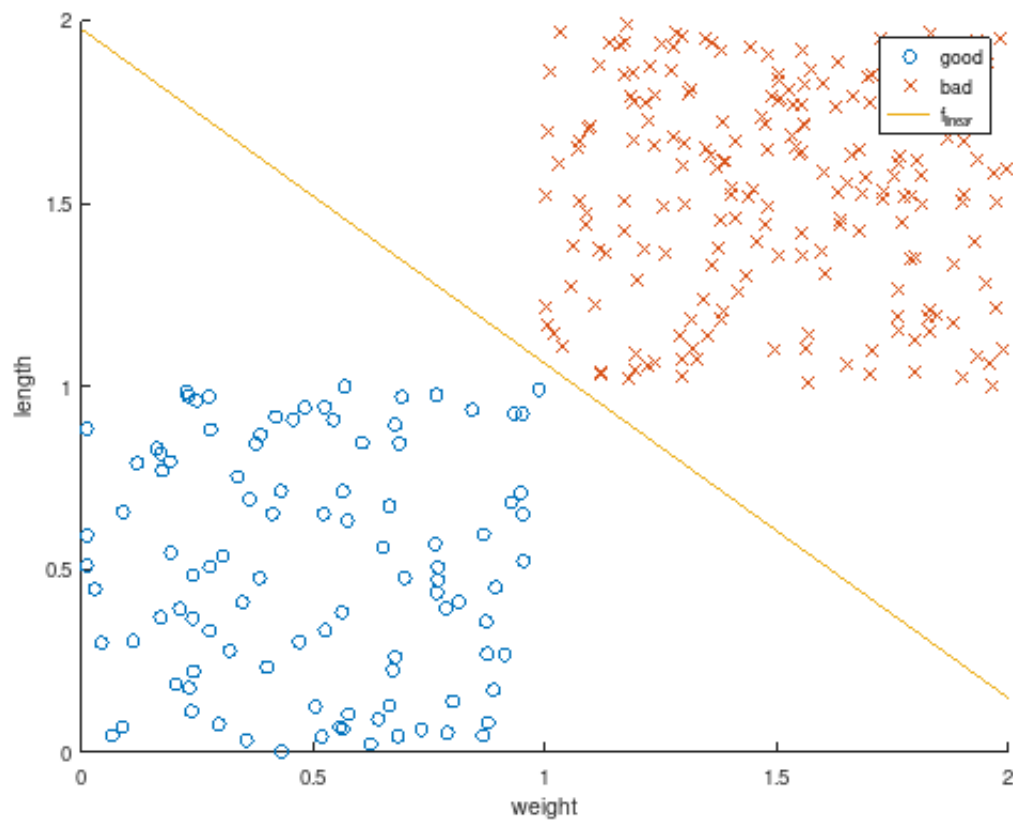
exitflag = 0
```

The computed solution x are the sought coefficients for $f_{\text{linear}}(w_i, l_i)$. Plotting the resulting function shows that it perfectly discriminates the measurements (test data).

```
axis equal
scatter(w(1:k), l(1:k), 'o'); % good
hold on;
scatter(w(k+1:m), l(k+1:m), 'x'); % bad

f_quad = @(w,l) 1.*x(1) + w.*x(2) + l.*x(3);
ezplot(f_quad, [0,2,0,2]);

title('');
xlabel('weight');
ylabel('length');
legend({'good', 'bad', 'f_{linear}'});
```



After this “training” the linear discrimination function $f_{\text{linear}}(w_i, l_i)$ can be used to classify other data pairs as well.

19.1 Non-linear classification

See also *exercise LP02* [BV04] (chapter 8.6).

19.1.1 Creation of labeled measurements (“training data”)

Create m random data pairs (w_i, l_i) of weights and lengths.

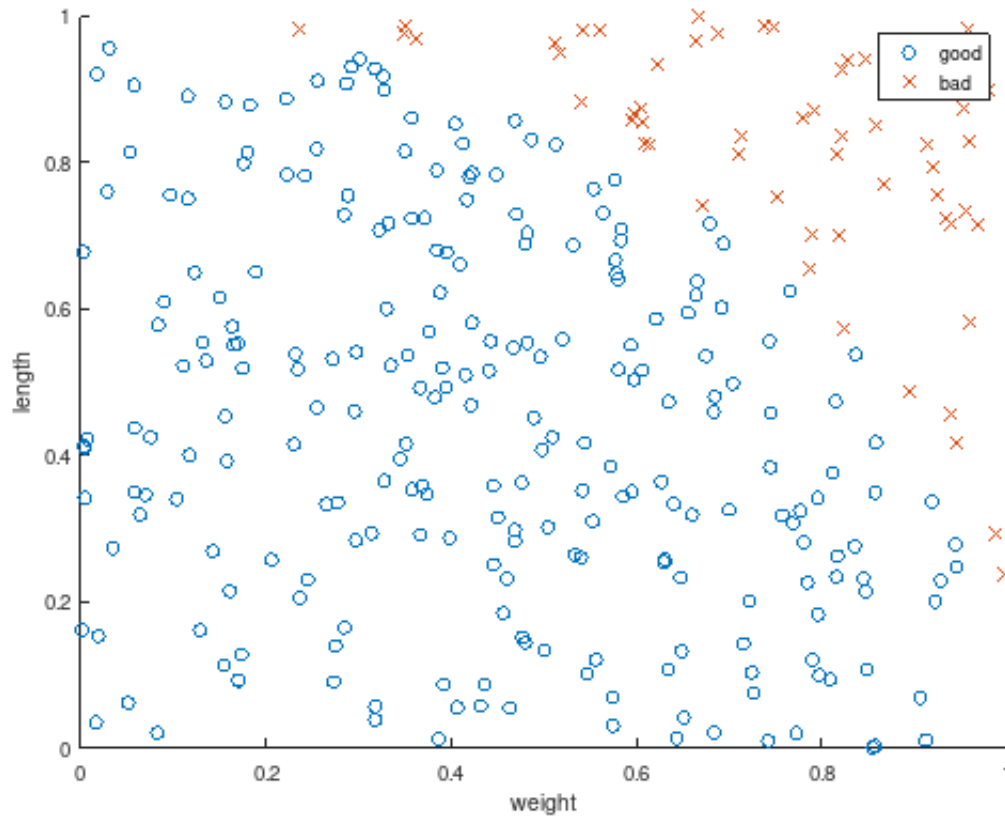
```
m = 300;  
w = rand(m,1); % weight  
l = rand(m,1); % length
```

Split the measurements into two sets according to a non-linear function (in this case an unit circle), such that a data pair (w_i, l_i) is considered as “good” for $1 \leq i \leq k$ and “bad” otherwise.

```
idx = sqrt (w.^2 + l.^2) < 1;  
k = sum(idx);  
w = [w(idx); w(~idx)];  
l = [l(idx); l(~idx)];
```

Plot the resulting data to discriminate.

```
axis equal  
scatter(w(1:k), l(1:k), 'o'); % good  
hold on;  
scatter(w(k+1:m), l(k+1:m), 'x'); % bad  
xlabel('weight');  
ylabel('length');  
legend({'good', 'bad'});
```



19.1.2 Classification

It is obvious, that linear discrimination fails. There exists no linear function

$$f_{\text{linear}}(w_i, l_i) = x_0 + \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} w_i \\ l_i \end{pmatrix} = x_0 + x_1 \cdot w_i + x_2 \cdot l_i$$

to separate

- the “good” blue circles: $f_{\text{linear}}(w_i, l_i) \leq -1$ for $1 \leq i \leq k \leq m$ and
- the “bad” red crosses: $f_{\text{linear}}(w_i, l_i) \geq 1$ for $k < i \leq m$.

However, using a quadratic discrimination function will work in this example:

$$\begin{aligned} f_{\text{quad}}(w_i, l_i) &= x_0 + \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} w_i \\ l_i \end{pmatrix} + \begin{pmatrix} w_i \\ l_i \end{pmatrix}^T \begin{pmatrix} x_4 & x_3 \\ x_3 & x_5 \end{pmatrix} \begin{pmatrix} w_i \\ l_i \end{pmatrix} \\ &= x_0 + x_1 \cdot w_i + x_2 \cdot l_i + 2x_3 \cdot w_i \cdot l_i + x_4 \cdot w_i^2 + x_5 \cdot l_i^2 \end{aligned}$$

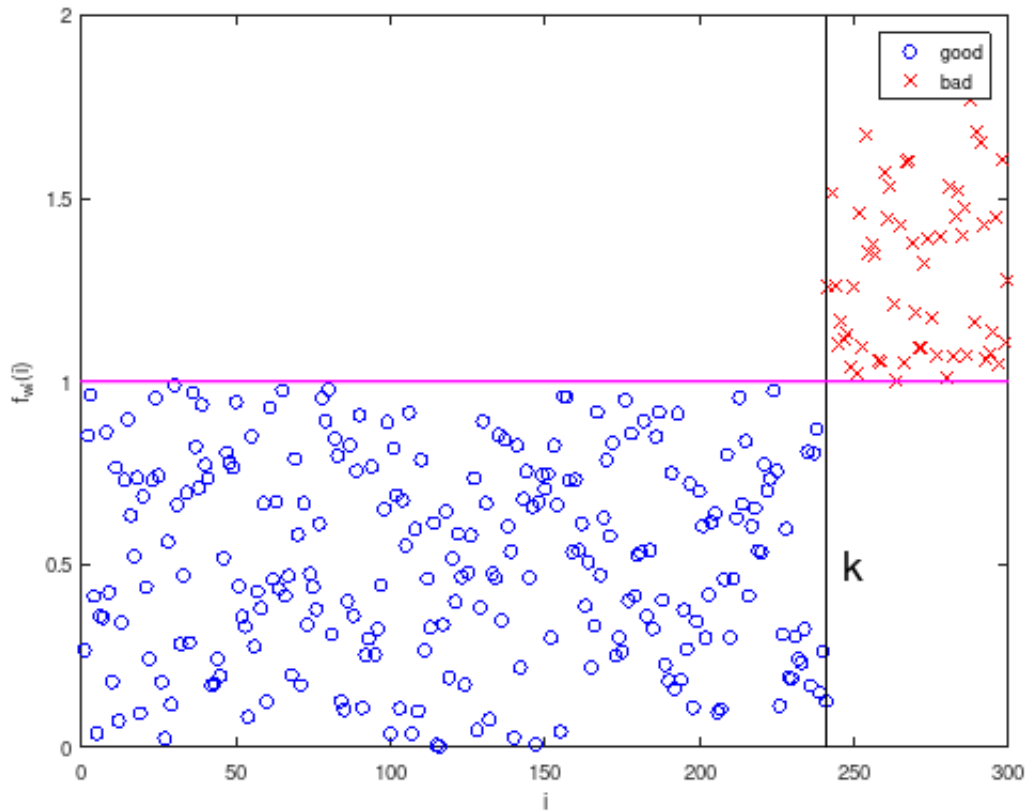
Roughly speaking, the quadratic discrimination function maps the data into higher dimensions (**expanded measurement vector**) and projects it to a scalar value. The optimization problem becomes finding a separating hyperplane in the expanded measurement vector, which is a linear classifier with the parameters x_0 to x_5 . This technique has been used in extending **neural network models for pattern recognition**.

To illustrate the approach, consider the squared sum of the test data’ s weights and lengths:

$$f_{\text{wt}}(i) = w_i^2 + l_i^2,$$

with $1 \leq i \leq m$.

```
axis equal
plot (1:k, w(1:k) .^ 2 + l(1:k) .^ 2, 'bo'); % good
hold on;
plot (k+1:m, w(k+1:m) .^ 2 + l(k+1:m) .^ 2, 'rx'); % bad
plot ([k,k],[0,2], 'k', 'LineWidth', 1);
plot ([0,m],[1,1], 'm', 'LineWidth', 2);
text (k + 5, 0.5, 'k', 'FontSize', 20);
xlabel('i')
ylabel('f_{wl}(i)')
legend({'good', 'bad'})
```



Regarding the construction of the data set above, it is no surprise that $f_{wl}(i) = w_i^2 + l_i^2 = 1$ is a separating hyperplane.

19.1.3 Feasibility problem as Linear Program (LP)

For now let's pretend, that the structure of the measurements (test data) is not known, and a general quadratic classifier is sought.

Like in [exercise LP02](#) this optimization problem can be formulated as LP on the expanded measurement vector:

$$\begin{aligned} & \text{minimize} && (0 \ 0 \ 0 \ 0 \ 0 \ 0) \ x \\ & \text{subject to} && \begin{pmatrix} 1 & w_i & l_i & 2w_i l_i & w_i^2 & l_i^2 \end{pmatrix} x \leq -1, \quad i = 1, \dots, k, \\ & && -\begin{pmatrix} 1 & w_i & l_i & 2w_i l_i & w_i^2 & l_i^2 \end{pmatrix} x \leq -1, \quad i = k+1, \dots, m, \end{aligned}$$

```

c = [0 0 0 0 0 0];
A = [ ones(k, 1), w(1:k), l(1:k), 2.*w(1:k).*l(1:k), w(1:k).^2, l(1:k).^2; ...
      -ones(m-k, 1), -w(k+1:m), -l(k+1:m), -2.*w(k+1:m).*l(k+1:m), -w(k+1:m).^2, -l(k+1:m).^2 ];
b = -ones(m, 1);
Aeq = []; % No equality constraints
beq = [];
lb = -inf(6, 1); % x0 to x5 are free variables
ub = inf(6, 1);
CTYPE = repmat('U', m, 1); % Octave: A(i,:)*x <= b(i)
x0 = []; % default start value
[x,~,exitflag] = linprog(c,A,b,Aeq,beq,lb,ub,x0); % Matlab: exitflag=1 success
[x,~,exitflag] = glpk(c,A,b,lb,ub,CTYPE)          % Octave: exitflag=0 success

```

```

x =

    6.6295
   -113.1690
   -171.7719
    57.0760
   115.8150
   162.4596

exitflag = 0

```

The computed solution differs from the unit circle, but nevertheless perfectly discriminates the test data and a quadratic classifier is found without prior knowledge of the test data structure.

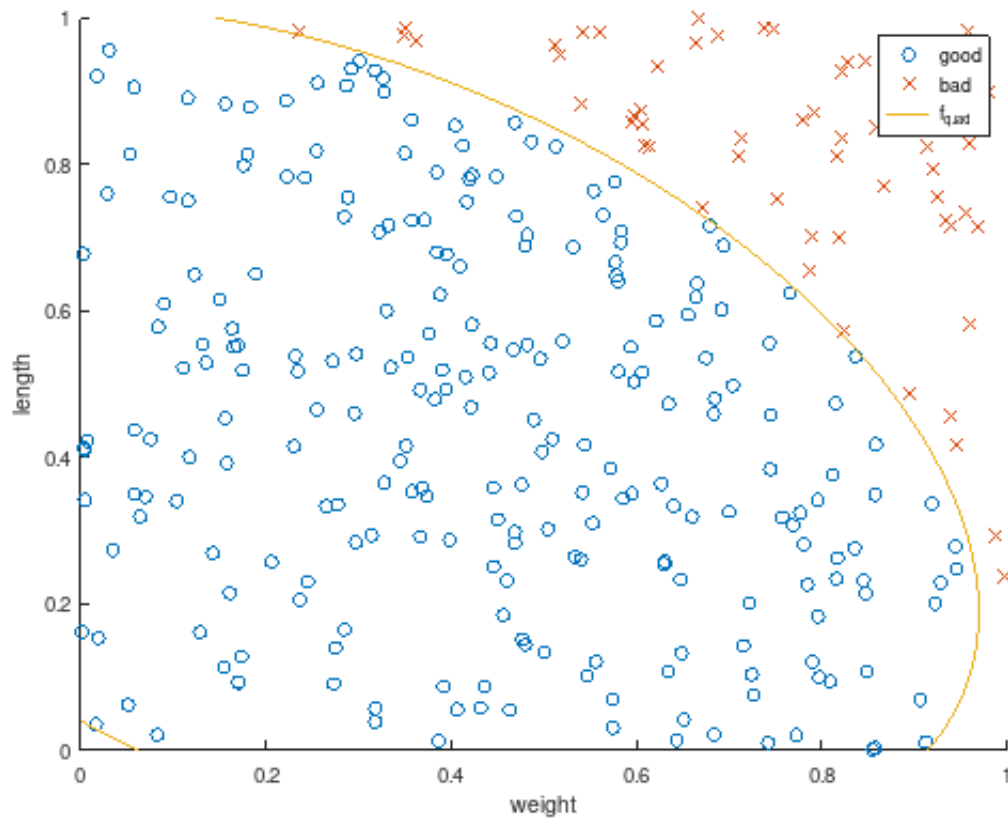
```

axis equal
scatter(w(1:k), l(1:k), 'o'); % good
hold on;
scatter(w(k+1:m), l(k+1:m), 'x'); % bad

f_quad = @(w,l) 1.*x(1) + w.*x(2) + l.*x(3) + 2.*w.*l.*x(4) + w.^2.*x(5) + l.^2.*x(6);
ezplot(f_quad, [0,1,0,1]);

title('');
xlabel('weight');
ylabel('length');
legend({'good', 'bad', 'f_{quad}'});

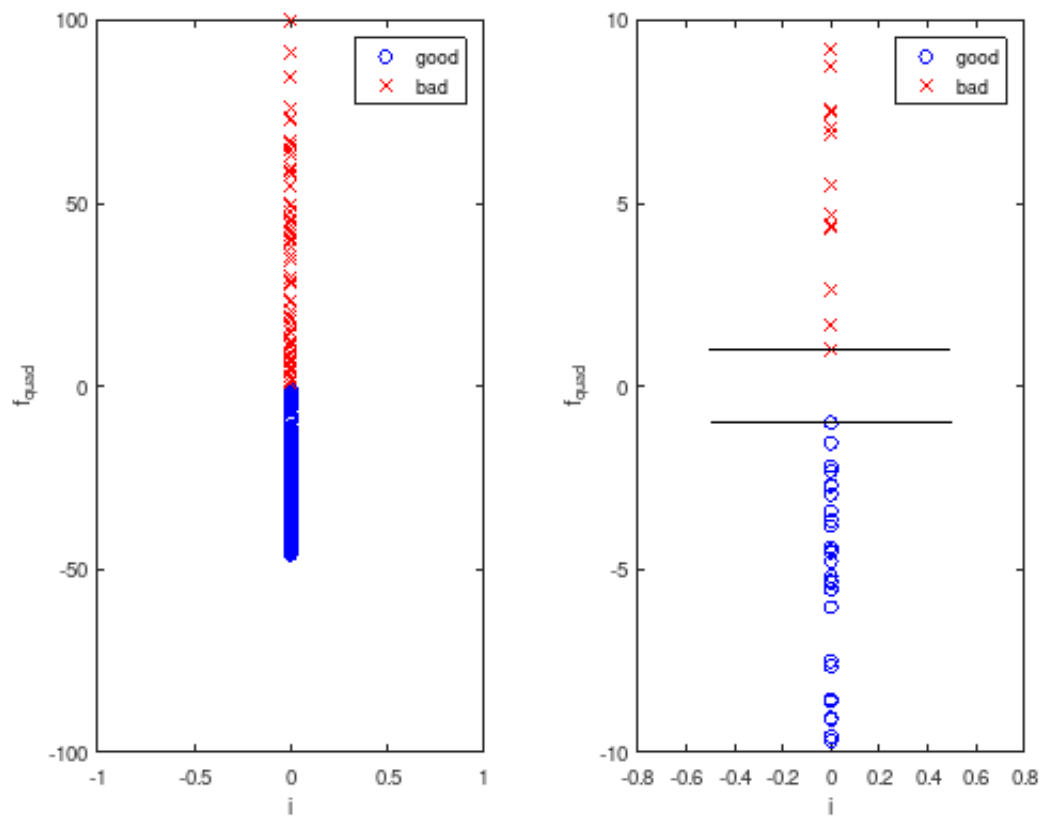
```



The quadratic discrimination function $f_{quad}(w, l)$ maps all data into higher dimensions and projects it to a scalar value to distinguish “good” and “bad” data pairs.

```
subplot (1, 2, 1)
plot (zeros(k,1), f_quad(w(1:k), l(1:k)), 'bo'); % good
hold on;
plot (zeros(m-k,1), f_quad(w(k+1:m), l(k+1:m)), 'rx'); % bad
xlabel('i')
ylabel('f_{quad}')
legend ({'good', 'bad'})
ylim ([-100 100]);

subplot (1, 2, 2)
plot (zeros(k,1), f_quad(w(1:k), l(1:k)), 'bo'); % good
hold on;
plot (zeros(m-k,1), f_quad(w(k+1:m), l(k+1:m)), 'rx'); % bad
plot ([-0.5, 0.5], [1, 1], 'k');
plot ([-0.5, 0.5], [-1, 1], 'k');
xlabel('i')
ylabel('f_{quad}')
legend ({'good', 'bad'})
ylim ([-10 10]);
```



After this “training” the discrimination function can be used to classify other data pairs as well.

MO01 - THE ILLUMINATION PROBLEM

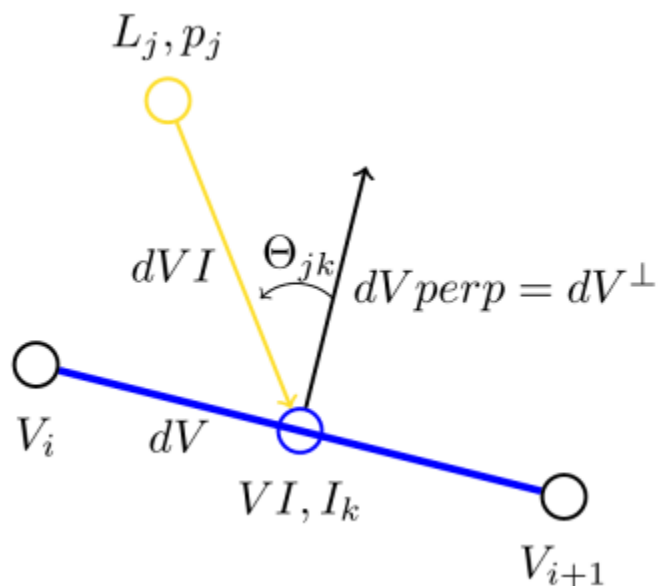
Source: <https://see.stanford.edu/Course/EE364A> (hw3extra.pdf task 3).

A physical law from photometry states

$$I = \frac{p}{r^2} \cos(\Theta),$$

the **illuminance** on a patch I (SI unit: lux [$lx = \frac{lm}{m^2}$]) depends on:

1. the **luminous intensity** p (SI unit: candela [$cd = \frac{lm}{sr}$]) of the lamps,
2. the distance to the respective patch r (SI unit: meter [m]),
3. the angle Θ to between the lamp and patch.



In case of the given problem data, there are $m = 10$ lamps $L_j, j = 1, \dots, m$, with assigned **luminous intensities** p_j and $n = 20$ patches with respective **illumination values** $I_k, k = 1, \dots, n$.

In the following the SI units are neglected. The following linear dependencies between I_k and p_j hold:

$$I_k = \underbrace{\frac{1}{r_{kj}^2} \max(\cos(\Theta_{kj}), 0)}_{a_{kj}} p_j$$

with $r_{kj} = \|dVI\|$ and

$$\cos(\Theta_{kj}) = \frac{(dVI)^T(dV_{perp})}{\|dVI\| \cdot \|dV_{perp}\|}$$

To regard all lamps p_j , the **luminous intensities** have to be summed up:

$$I_k = \sum_{j=1}^m a_{kj} p_j.$$

Regarding all $n = 20$ patches, the following overdetermined linear system of equation $I = A \cdot p$ follows:

$$\begin{pmatrix} I_1 \\ \vdots \\ I_n \end{pmatrix} = \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{pmatrix} \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix}.$$

As ultimate goal, all patch illuminations I_k should be as close as possible to the **desired illumination** I_{des} .

Finally, the following constrained non-linear optimization problem can be formulated:

$$\begin{aligned} & \text{minimize} && \max_{k=1,\dots,n} |\log(I_k) - \log(I_{des})| \\ & \text{subject to} && 0 \leq p_j \leq p_{max} \quad j = 1, \dots, m. \end{aligned}$$

The objective function formulates a quality measure $f_{optimal}(p)$ depending on the **luminous intensity** of the lamps p_j , that minimizes the maximal logarithmic distance between the **actual patch illumination** $I_k = A_{(k,:)}p$ and the **desired patch illumination** I_{des} .

$$\begin{aligned} f_{optimal}(p, I_{des}) &:= \max_{k=1,\dots,n} |\log(I_k) - \log(I_{des})| \\ &= \max_{k=1,\dots,n} |\log(A_{(k,:)}p) - \log(I_{des})| \end{aligned}$$

Last but not least, the constraints might seem trivial, but are for a practical application of importance: the **luminous intensity** of a lamp p_j has the physical constraints that it cannot “shine negative” and it cannot “shine brighter than possible”, $0 \leq p_j \leq p_{max}$.

```
L = [linspace(0,1,10);
      1.9, 1.8, 1.0, 1.1, 1.9, 1.8, 1.9, 1.7, 1.5, 1.5];

m = size(L, 2); % number of lamps

% begin and endpoints of patches
V = [linspace(0,1,21);
      .4*[0.0, 0.1, 0.15, 0.2, 0.1, 0.2, 0.3, 0.0, 0.0, 0.0, ...
          0.1, 0.2, 0.2, 0.0, 0.1, 0.05, 0.1, 0.1, 0.0, 0.2, 0.1]];
n = size(V, 2) - 1; % number of patches

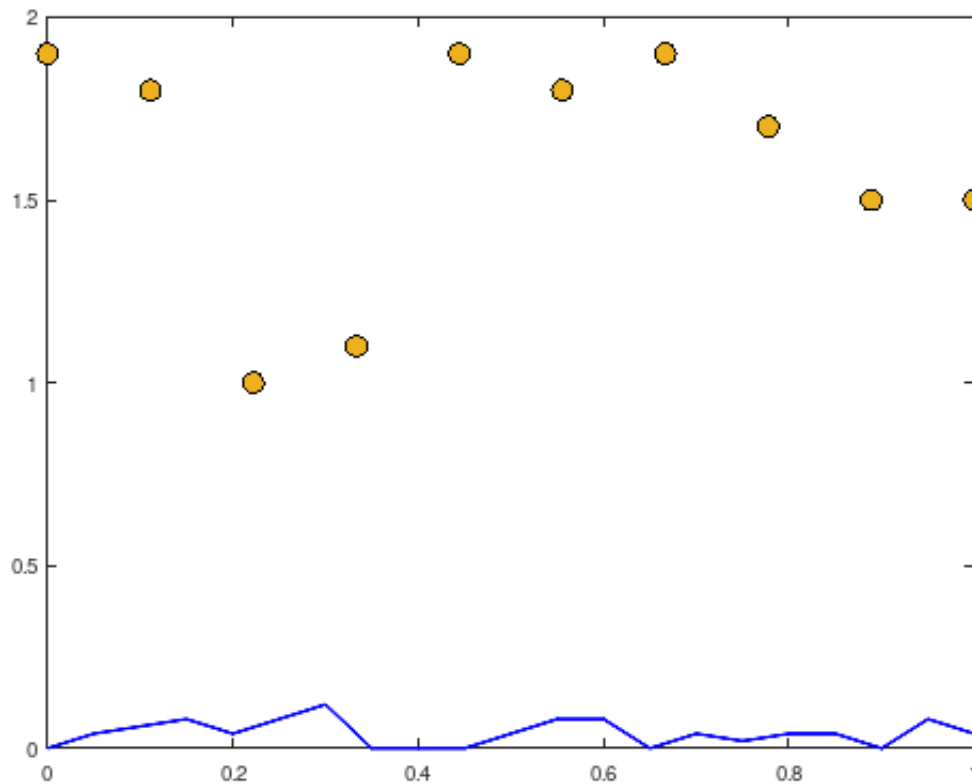
plot(L(1,:), L(2,:), 'ko', 'MarkerSize', 10, ...
      'MarkerFaceColor', [0.9290 0.6940 0.1250]);
hold on;
plot(V(1,:), V(2,:), 'b-', 'LineWidth', 2);

% construct A
```

(continues on next page)

(continued from previous page)

```
warning('off', 'Octave:colon-nonscalar-argument');
dV = V(:,2:n+1) - V(:,1:n); % tangent to patches
VI = V(:,1:n) + .5*dV;      % midpoint of patches
A = zeros (n, m);
for i = 1:n
    for j = 1:m
        dVI = L(:,j) - VI(:,i);
        dVperp = null (dV(:,i)'); % upward pointing normal
        if (dVperp(2) < 0)
            dVperp = -dVperp;
        end
        A(i,j) = max (0, dVI'*dVperp/...
            (norm(dVI)*norm(dVperp)))./norm(dVI)^2;
    end
end
```



Read task 3 from [hw3extra.pdf](#) and try to find an optimal solution with

1. Uniform p_j
2. Least-squares
3. Linear Programming
4. Convex Programming (<https://yalmip.github.io/> or <http://cvxr.com/cvx/>).

Solution hints.

Part III

Back matter

CODE LISTINGS

21.1 newton_simple.m

```
function [x, fval, exitflag] = newton_simple (fun, x0, options)
% Find minimum of unconstrained multivariable function.
%
% Input:
%     fun - function (returning function value, gradient
%               vector value and Hessian matrix value)
%     x0 - initial guess
%     options - structure with fields
%         .MaxIterations - max. number of iterations
%         .OptimalityTolerance - tolerance for the norm of gradient
%         .Display - display details for 'iter'
%
% Output:
%     x - solution
%     fval - objective value at solution
%     exitflag - 0 = no solution found
%               1 = solution found
%
marginchk(2,3);

opt = struct ();
if ((nargin == 3) && isstruct (options))
    opt = options;
end
if (~isfield (opt, 'MaxIterations') || isempty (opt.MaxIterations))
    opt.MaxIterations = 30;
end
if (~isfield (opt, 'Display'))
    opt.Display = 'off';
end
if (~isfield (opt, 'OptimalityTolerance'))
    opt.OptimalityTolerance = 1e-9;
end
if (strcmp(opt.Display, 'iter'))
    fprintf('Iteration\tStep-size\tf(x)\t\t\t|df(x)|\n')
end

exitflag = 0;
x = x0(:);
```

(continues on next page)

(continued from previous page)

```
i = 0;
while (i < opt.MaxIterations)
    [fval, fgrad, fhess] = fun(x);
    if (norm (fgrad, 'inf') < opt.OptimalityTolerance)
        exitflag = 1; % success!
        return;
    end
    d = fhess \ -fgrad;
    if (strcmp (opt.Display, 'iter'))
        fprintf('%5d\t\t%e\t%e\t%e\n', i, norm (d, 'inf'), ...
            fval, norm (fgrad, 'inf'));
    end
    x = x + d;
    i = i + 1;
end

if (i >= opt.MaxIterations)
    warning ('MaxIterations = %d reached.\n', opt.MaxIterations);
end
end
```

21.2 nelder_mead.m

```
function [x, fval, exitflag] = nelder_mead (fun, x0, options)
% Find minimum of unconstrained multivariable function.
%
% Input:
%     fun - function
%     x0 - initial guess
%     options - structure with fields
%         .MaxIterations - max. number of iterations
%         .ToIFun        - tolerance for function values
%         .Display        - display details for 'iter',
%                           display animation for 'full'.
%
% Output:
%     x - solution
%     fval - objective value at solution
%     exitflag - 0 = no solution found
%               1 = solution found
%
narginchk(2,3);

opt = struct ();
if ((nargin == 3) && isstruct (options))
    opt = options;
end
if (~isfield (opt, 'MaxIterations') || isempty (opt.MaxIterations))
    opt.MaxIterations = 30;
end
if (~isfield (opt, 'Display'))
    opt.Display = 'off';
end
```

(continues on next page)

(continued from previous page)

```

end
if (~isfield (opt, 'TolFun'))
    opt.TolFun = 1e-9;
end
if (strcmp(opt.Display, 'iter') || strcmp (opt.Display, 'full'))
    disp (' Iteration      Func-count      min f(x)          Procedure');
    h1 = [];
    h2 = [];
    h3 = [];
end

exitflag = 0;

% Parameter, here constant.
alpha = 1;
beta = 2;
gamma = 0.5;
len = 1;

% Create start simplex.

n = length (x0);
p = (len / sqrt (2)) * (sqrt (n + 1) - 1) / n;
q = [zeros(n,1), ones(n) * p];
x = repmat (x0(:), 1, n + 1) + q ...
    + [zeros(n,1), (len / sqrt (2)) * eye(n)];

% Compute all function values.

fvals = zeros (1, n + 1);
for i = 1:(n + 1)
    fvals(i) = fun(x(:,i));
end
fcount = n + 1;

% Sort simplex (left is best).

[fvals, idx] = sort (fvals);
x = x(:,idx);

i = 0;
while (i < opt.MaxIterations)

    % Compute centroid.
    m = sum(x(:,1:n), 2) / n;

    % Compute reflect.
    r = m + alpha * (m - x(:,end));
    fr = fun(r);
    fcount = fcount + 1;

    if (strcmp (opt.Display, 'full'))
        delete (h1);
        delete (h2);
        delete (h3);
        h1 = plot3 ([x(1,:), x(1,1)], [x(2,:), x(2,1)], [fvals, fvals(1)]);
    end
end

```

(continues on next page)

(continued from previous page)

```

h2 = plot3 (x(1,1), x(2,1), fvals(1), 'ro');
h3 = plot3 ([x(1,end), r(1)], [x(2,end), r(2)], [fvals(end), fr], 'ro');
pause (0.5);
end

if (fr <= fvals(2)) % Case 1: reflect

    if (fr < fvals(1)) % Case 2: expand

        e = m + beta * (m - x(:,end));
        fe = fun(e);
        fcount = fcount + 1;

        if (fe < fr) % Expansion successful =)
            r = e;
            fr = fe;
        end

        % Replace worst x(:,end) with r (or s).
        x = [r, x(:,1:n)];
        fvals = [fr, fvals(1:n)];

        step_method = 'expand';

    else

        % Replace worst x(:,end) with r.
        x = [x(:,1), r, x(:,2:n)];
        fvals = [fvals(1), fr, fvals(2:n)];

        step_method = 'reflect';

    end

else % Case 3: Contract / Shrink

    if (fr > fvals(end))
        step_method = 'contract inside';
        c = m - gamma * (m - x(:,end));
    else
        step_method = 'contract outside';
        c = m + gamma * (m - x(:,end));
    end

    fc = fun(c);
    fcount = fcount + 1;

    if (fc < fvals(end))

        % Replace worst x(:,end) with c.
        x = [x(:,1:n), c];
        fvals = [fvals(1:n), fc];

    else

        step_method = 'shrink';
    end

```

(continues on next page)

(continued from previous page)

```

    x = (x + repmat (x(:,1), 1, n + 1)) / 2;

    % Compute new function values again.
    for i = 1:(n + 1)
        fvals(i) = fun(x(:,i));
    end
    fcount = fcount + n + 1;

end

% Sort again.
[fvals, idx] = sort (fvals);
x = x(:,idx);

end

fval = fvals(1);

if (strcmp (opt.Display, 'iter') || strcmp (opt.Display, 'full'))
    fprintf ('      %2d      %2d      %+8f      %s\n', ...
        i, fcount, fval, step_method);
end

% Check stopping criteria.
if (var (fvals) < opt.TolFun)
    x = x(:,1)';
    exitflag = 1; % success!
    return;
end

i = i + 1;

end

x = x(:,1)';

if (i >= opt.MaxIterations)
    warning ('MaxIterations = %d reached.\n', opt.MaxIterations);
end
end
end

```

21.3 lp_solver.m

```

function [x, fval, exitflag, output] = lp_solver (c, A, b, x0, options)
% LP_SOLVER Solver for linear programs (LP).
%
%      Input:
%
%      min   c'*x
%      s.t.  A*x = b
%            x >= 0
%

```

(continues on next page)

(continued from previous page)

```

%      x0 - initial guess
%
%      Output:
%      x - solution
%      fval - objective value at solution
%      exitflag - 0 = no solution found
%               1 = solution found
%      output - struct with fields:
%               .xpath - history of x
%
%
marginchk (3, 5);

opt = struct ();
if ((margin == 5) && isstruct (options))
    opt = options;
end
if (~isfield (opt, 'MaxIterations') || isempty (opt.MaxIterations))
    opt.MaxIterations = 10;
end
if (~isfield (opt, 'Display'))
    opt.Display = 'off';
end
if (~isfield (opt, 'OptimalityTolerance'))
    opt.OptimalityTolerance = 1e-8;
end

c = c(:);
b = b(:);
n = length (c);
m = length (b);

% Hessian matrix for Newton step.
H = @(x,s) [ zeros(n), -eye(n),          A'; ...
             diag(s),  diag(x), zeros(n, m); ...
             A,        zeros(m,n + m) ];

% Gradient vector for Newton step.
G = @(x,s,y,mu) [ c - s + A' * y; ...
                  x .* s - mu * ones(n, 1); ...
                  A * x - b ];

% Initial values. x should be an inner LP point.
if ((margin > 3) && isempty (x0))
    x = x0(:);
else
    x = ones (n, 1);
end
s = ones (n, 1);
y = ones (m, 1);
mu = 0.1;

% Track central path.
output.xpath = x;

if (strcmp (opt.Display, 'iter'))

```

(continues on next page)

(continued from previous page)

```

    fprintf('Iteration   fval       ||dx||       ||dy||       ||ds||\n')
end

fval = c'*x;
exitflag = 0;

% Newton iteration.
for i = 1:opt.MaxIterations

    % Newton step on optimality conditions.
    d = H(x,s) \ -G(x,s,y,mu);

    % Stop iteration if x does not change any more.
    if (norm(d(1:n), 'inf') < opt.OptimalityTolerance)
        if (strcmp(opt.Display, 'iter'))
            fprintf('\nIPM converged in step %d.\n\n', i);
        end
        exitflag = 1;
        break;
    end

    % Update variables.
    x = x + d(1:n);
    s = s + d((1:n) + n);
    y = y + d((1:m) + (n * 2));

    fval = c'*x;

    mu = mu^2; % Shrink mu fast.

    % Output step statistics.
    output.xpath(:,end+1) = x;
    if (strcmp(opt.Display, 'iter'))
        fprintf('%5d      %.2f   %.2e   %.2e   %.2e\n', ...
            i, fval, ...
            norm(d(1:n), 'inf'), ...
            norm(d((1:m) + n), 'inf'), ...
            norm(d((1:m) + (n * 2)), 'inf'));
    end
end

if (i == opt.MaxIterations)
    warning('MaxIterations = %d reached.\n', opt.MaxIterations);
end

end

```

21.4 um01_experiment.m

Requires *newton_simple* and *nelder_mead* function.

```
function um01_experiment ()

disp ('Step 1: plotting the problem')

f = @(x,y) x.^4 + 2.*x.*y + (1 + y).^2;

[x, y] = meshgrid (linspace (-3, 3, 20));
mesh (x, y, f(x,y));
grid on;
xlabel ('x');
ylabel ('y');
zlabel ('f(x,y)');
title ('f(x,y) = x^4 + 2xy + (1 + y)^2');
hold on;
plot3 (1, -2, -2, 'ro');

view (-2, 70);

disp ('Step 2: Try fminsearch')

format long;
xpath = zeros(0, 3); % Memorize path

x0 = [1, 1];
options.Display = 'iter';
[xopt, fval, exitflag] = fminsearch (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'bo-'});

disp ('Step 3: Try fminunc')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
options.GradObj = 'on';
[xopt, fval, exitflag] = fminunc (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);
```

(continues on next page)

(continued from previous page)

```

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'go-'});

disp ('Step 4: Try newton_simple')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
[xopt, fval, exitflag] = newton_simple (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'ro-'});

disp ('Step 5: Try nelder_mead')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
options.MaxIterations = 30;
[xopt, fval, exitflag] = nelder_mead (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'mo-'});

% Nested function to pass to fminsearch and fminunc.

function [fx, gx, hx] = fun (x)
    fx = x(1).^4 + 2.*x(1).*x(2) + (1 + x(2)).^2;

    gx = [ 4.*x(1).^3 + 2.*x(2); ...
          2.*x(1) + 2.*(1 + x(2)) ];

    hx = [ 12.*x(1).^2, 2; ...
          2, 2 ];

```

(continues on next page)

(continued from previous page)

```

    xpath = [xpath; x(:)', fx]; % Memorize path
end

end

function animate_path (xpath, LineSpec)
    for i = 2:size (xpath, 1)
        plot3 (xpath(i-1:i,1), xpath(i-1:i,2), xpath(i-1:i,3), LineSpec{:});
        pause (0.1);
        drawnow ();
    end
end
end

```

21.5 um02_experiment.m

Requires *newton_simple* and *nelder_mead* function.

```

function um02_experiment ()

disp ('Step 1: plotting the problem')

f = @(x,y) x.^3 - 6.*x.*y + 8.*y.^3;

[x, y] = meshgrid (linspace (-2, 2, 20));
mesh (x, y, f(x,y));
grid on;
xlabel ('x');
ylabel ('y');
zlabel ('f(x,y)');
title ('f(x,y) = x^3 - 6xy + 8y^3');
hold on;
plot3 (1, 0.5, -1, 'ro');

view (-60, 50);

disp ('Step 2: Try fminsearch')

format long;
xpath = zeros(0, 3); % Memorize path

x0 = [1, 1];
options.Display = 'iter';
[xopt, fval, exitflag] = fminsearch (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

```

(continues on next page)

(continued from previous page)

```

animate_path (xpath, {'bo-'});

disp ('Step 3: Try fminunc')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
options.GradObj = 'on';
[xopt, fval, exitflag] = fminunc (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'go-'});

disp ('Step 4: Try newton_simple')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
[xopt, fval, exitflag] = newton_simple (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'ro-'});

disp ('Step 5: Try nelder_mead')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
options.MaxIterations = 4;
[xopt, fval, exitflag] = nelder_mead (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

```

(continues on next page)

(continued from previous page)

```

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'mo-'});

% Nested function to pass to fminsearch and fminunc.

function [fx, gx, hx] = fun (x)
    fx = x(1).^3 - 6.*x(1).*x(2) + 8.*x(2).^3;

    gx = [ 3.*x(1).^2 - 6.*x(2); ...
           24.*x(2).^2 - 6.*x(1) ];

    hx = [ 6.*x(1),      -6; ...
           -6, 48.*x(2) ];

    xpath = [xpath; x(:)', fx]; % Memorize path
end

end

function animate_path (xpath, LineSpec)
    for i = 2:size (xpath, 1)
        plot3 (xpath(i-1:i,1), xpath(i-1:i,2), xpath(i-1:i,3), LineSpec{:});
        pause (0.1);
        drawnow ();
    end
end
end

```

21.6 um03_experiment.m

Requires *newton_simple* and *nelder_mead* function.

```

function um03_experiment ()

disp ('Step 1: plotting the problem')

f = @(x,y) 100 * (y - x.^2).^2 + (1 - x).^2;

[x, y] = meshgrid (linspace (-2, 2, 30), linspace (-1, 4, 30));
mesh (x, y, f(x,y));
grid on;
xlabel ('x');
ylabel ('y');
zlabel ('f(x,y)');
title ('Rosenbrock function');
hold on;
plot3 (1, 1, 0, 'ro');

view (-30, 50);

disp ('Step 2: Try fminsearch')

```

(continues on next page)

(continued from previous page)

```

format long;
xpath = zeros(0, 3); % Memorize path

x0 = [-0.5, -0.5];
options.Display = 'iter';
[xopt, fval, exitflag] = fminsearch (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'bo-'});

disp ('Step 3: Try fminunc')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
options.GradObj = 'on';
[xopt, fval, exitflag] = fminunc (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'go-'});

disp ('Step 4: Try newton_simple')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
[xopt, fval, exitflag] = newton_simple (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'ro-'});

```

(continues on next page)

(continued from previous page)

```

disp ('Step 5: Try nelder_mead')

xpath = zeros(0, 3); % Memorize path

%x0 = [1, 1];
options.Display = 'iter';
options.MaxIterations = 4;
[xopt, fval, exitflag] = nelder_mead (@fun, x0, options);

disp ('Solution');
disp (xopt);

disp ('Objective value at solution');
disp (fval);

fprintf ('exitflag = %d\n', exitflag);

animate_path (xpath, {'mo-'});

% Nested function to pass to fminsearch and fminunc.
function [fx, gx, hx] = fun (x)
    fx = 100 * (x(2) - x(1).^2).^2 + (1 - x(1)).^2;

    gx = [-400 * x(1) * (x(2) - x(1).^2) - 2 * (1 - x(1)); ...
          200 * (x(2) - x(1).^2)];

    hx = [(1200 * x(1).^2 - 400 * x(2) + 2), -400 * x(1); ...
          -400 * x(1), 200];

    xpath = [xpath; x(:)', fx]; % Memorize path
end

end

function animate_path (xpath, LineSpec)
    for i = 2:size (xpath, 1)
        plot3 (xpath(i-1:i,1), xpath(i-1:i,2), xpath(i-1:i,3), LineSpec{:});
        pause (0.1);
        drawnow ();
    end
end
end

```

BIBLIOGRAPHY

BIBLIOGRAPHY

- [Ber16] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, third edition, 2016. ISBN 978-1-886529-05-2. URL: <http://www.athenasc.com/nonlinbook.html>.
- [BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN 978-0-521-83378-3. URL: <https://web.stanford.edu/~boyd/cvxbook/>.
- [CT17] Richard W. Cottle and Mukund N. Thapa. *Linear and Nonlinear Optimization*. International Series in Operations Research & Management Science. Springer, 2017. ISBN 978-1-4939-7053-7. doi:10.1007/978-1-4939-7055-1.
- [GNS08] Igor Griva, Stephen G. Nash, and Ariela Sofer. *Linear and Nonlinear Optimization*. SIAM, second edition, 2008. ISBN 978-0-89871-661-0. URL: <https://bookstore.siam.org/OT108/>.
- [Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373-395, 1984. doi:10.1007/BF02579150.
- [LRWW98] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. *SIAM Journal on Optimization*, 9(1):112-147, 1998. doi:10.1137/S1052623496303470.
- [LY16] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. International Series in Operations Research & Management Science. Springer, fourth edition, 2016. ISBN 978-3-319-18841-6. URL: <http://www.athenasc.com/nonlinbook.html>.
- [NM65] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308-313, 1965. doi:10.1093/comjnl/7.4.308.
- [PCB02] C.J. Price, I.D. Coope, and D. Byatt. A Convergent Variant of the Nelder-Mead Algorithm. *Journal of Optimization Theory and Applications*, 113(1):5-19, 2002. doi:10.1023/A:1014849028575.