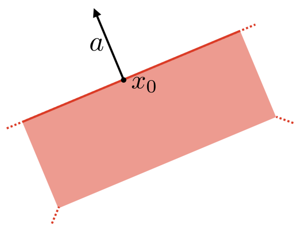
# 1 Convex Sets

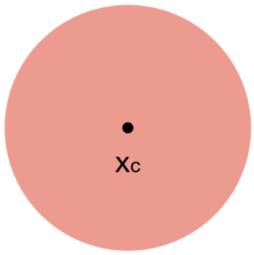
**1.1**. (a) Sketch the following sets $\Omega$ and (b) identify the convex sets:

- $\Omega = \{x | a^T x \leq b\}$
  *Solution:*

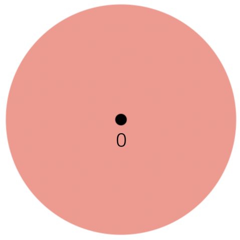
  Halfspace is a convex set.

- $\Omega = \{x | (x - x_c)^T P (x - x_c) \leq 1\}, P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
  *Solution:*

  It is a convex set.

- $\Omega = \{x | ||x||_2 \leq 1\}$
  *Solution:*

  It is a convex set.

**1.2**. Show that the solution set of linear equations $\{x|Ax = b\}$ with $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ is an affine set.

Solution: Suppose $x_1, x_2$ are in the solution set. We have $Ax_1 = b, Ax_2 = b$. For any w = $\alpha_1 A x_1 + \alpha_2 A x_2$, where $\alpha_1 + \alpha_2 = 1$, we can get $\alpha_1 A x_1 + \alpha_2 A x_2 = \alpha_1 b + \alpha_2 b = b$. So the solution set of the linear equations is an affine set.

## 2 Convex Functions

**2.1**. Define the function $f(x,y) = \frac{||Ax-b||_2^2}{Py-q}$ with $x,y \in \mathbb{R} \wedge Py-b > 0$. Check the convexity of $f$ over the domain.

*Solution:*

This function is the composition of the function $g(u,v) = u^T u/v$ with an affine transformation (u, v) = (Ax-b, Py-q). Therefore convexity of $f$ follows from the fact that $g$ is convex on $\{(u,v)|v>0\}$. For convexity of $g$ one can directly verify that if the Hessian is positive semi-definite. The gradient is

$$\nabla g(u,v) = \left[\frac{\partial g}{\partial u}, \frac{\partial g}{\partial v}\right]^T = \left[\frac{2u}{v}, -\frac{u^2}{v^2}\right]^T$$

The Hessian is

$$\nabla^2 g = \begin{bmatrix} \frac{\partial^2 g}{\partial u^2} & \frac{\partial^2 g}{\partial u \partial v} \\ \frac{\partial^2 g}{\partial u \partial v} & \frac{\partial^2 g}{\partial v^2} \end{bmatrix} = \begin{bmatrix} \frac{2}{v} & \frac{-2u}{v^2} \\ \frac{-2u}{v^2} & \frac{2u^2}{v^3} \end{bmatrix}.$$

We can easily verify that the Hessian is positive semi-definite when $v > 0$, since

$$(s,t)\nabla^2 g(s,t)^T = \frac{2}{v^3}(s,t)(v,-u)^T(v,-u)(s,t)^T = \frac{2}{v^3}(sv-ut)^2 \geq 0$$

for any $(s,t)$.

So, the function is convex when $Py-b > 0$.

## 3 Convex Problems

**3.1**. For the following optimization problem

$$\begin{aligned} \text{minimize} \quad & \|(2x+3y,-3x)^T\|_\infty \\ \text{subject to} \quad & |x-2y| \leq 3 \end{aligned}$$

(a) Express the problem as a a linear program. (b) Convert the LP so that all variables are in $\mathbb{R}_+$ and there are no other inequality constraints.

*Solution:*

(a) The linear program is as follows:

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & t \geq 2x+3y \\ & t \geq -2x-3y \\ & t \geq -3x \\ & t \geq 3x \\ & x-2y \leq 3 \\ & -x+2y \leq 3 \end{aligned}$$

(b) By introducing slack variables, we can eliminate inequality constraints.

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & -t+2x+3y+s_1=0 \\ & -t-2x-3y+s_2=0 \\ & -t-3x+s_3=0 \\ & -t+3x+s_4=0 \\ & x-2y-3+s_5=0 \\ & -x+2y-3+s_6=0 \\ & s_1,s_2,s_3,s_4,s_6,s_6 \geq 0 \end{aligned}$$

Furthermore, to make sure all variables are in $\mathbb{R}_+$, we need to add more slack variables. The problem is converted to:

$$\begin{aligned} \text{minimize} \quad & t^+-t^- \\ \text{subject to} \quad & -(t^+-t^-)+2(x^+-x^-)+3(y^+-y^-)+s_1=0 \\ & -(t^+-t^-)-2(x^+-x^-)-3(y^+-y^-)+s_2=0 \\ & -(t^+-t^-)-3(x^+-x^-)+s_3=0 \\ & -(t^+-t^-)+3(x^+-x^-)+s_4=0 \\ & (x^+-x^-)-2(y^+-y^-)-3+s_5=0 \\ & -(x^+-x^-)+2(y^+-y^-)-3+s_6=0 \\ & t^+,t^-,x^+,x^-,y^+,y^-,s_1,s_2,s_3,s_4,s_5,s_6 \geq 0 \end{aligned}$$

## 4 Duality

**4.1**. Derive a dual problem for

$$\text{minimize} \quad -\sum_{i=1}^{m} \log(b_i - a_i^T x)$$

with domain $\{x | a_i^T x < b_i, i = 1, ..., m\}$. First introduce new variables $y_i$ and equality constraints $y_i = b_i - a_i^T x$. Write out the infimum in the final result.

*Solution:*

We derive the dual of the problem

$$\begin{array}{ll} \text{minimize} & -\sum_{i=1}^{m} \log y_i \\ \text{subject to} & y = b - Ax, \end{array}$$

where $A \in \mathbb{R}^{m \times n}$ has $a_i^T$ as its $i$th row. The Lagrangian is

$$L(x, y, \nu) = -\sum_{i=1}^{m} \log y_i + \nu^T (y - b + Ax)$$

and the dual function is

$$g(\nu) = \inf_{x,y} (-\sum_{i=1}^{m} \log y_i + \nu^T (y - b + Ax))$$

The term $\nu^T Ax$ is unbounded below as a function of $x$ unless $A^T \nu = 0$. The terms in $y$ are unbounded below if $\nu \not\succ 0$, and achieve their minimum for $y_i = 1/\nu_i$ otherwise. We therefore find the dual function

$$g(\nu) = \begin{cases} \sum_{i=1}^{m} \log \nu_i + m - b^T \nu & A^T \nu = 0, \nu \succ 0 \\ -\infty & otherwise \end{cases}$$

and the dual problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{m} \log \nu_i + m - b^T \nu \\ \text{subject to} & A^T \nu = 0. \end{array}$$

# 5 Line Search

**5.1**. Consider the function

$$f(x,y) = 4(x-1/2)^2 + y^2.$$

A line search optimization algorithm starting at $(x^{(0)}, y^{(0)}) = (1,0)$ found a search direction $(\Delta x, \Delta y) = (-1,0)$ for which the function $f$ constrained to a line gives the function $\phi(t)$ in the plot.

(1) What is the expression for $\phi(t)$ for the give starting point and direction?

$$\begin{aligned}
\phi(t) &= f(x + t\Delta x) \\
&= f\left( \begin{pmatrix} 1 \\ 0 \end{pmatrix} + t \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right) \\
&= 4((1-t) - 1/2)^2 + (0 + t0)^2 \\
&= 4(1/2 - t)^2 \\
&= 4t^2 - 4t + 1
\end{aligned}$$

(2) Compute the step that an exact line search would produce for this function.

$$\begin{aligned}
\phi'(t) &= 0 \\
8t - 4 &= 0 \\
t^* &= 1/2
\end{aligned}$$

(3) Sketch the interval that a backtracking line search satisfying the armijo condition with $\alpha = 0.25$ would consider. Explain why, a geometric argument is enough.

At $t = 0$ the slope is $\phi'(0) = -4$, thus with $\alpha = 0.25$ the slope is $-1$, and all points under this line up to the intersection with $\phi$, call it $t_{max}$ are in the interval: $[0, t_{max}]$.

(4) Sketch the interval that a backtracking line search satisfying the Wolfe conditions with $\alpha = 0.25$ and $\beta = 0.8$ would consider. Explain why, a geometric argument is enough.

$\beta = 0.8 \Rightarrow -8 * 0.8 = -3.2$ is a lower bound for the slope now.

Tracing this to lines it is clear that the interval now is from around 0.1 to 0.74.

(5) Verify the intervals above by performing the computations.

The upper limit is the intersection of $\phi(t)$ with the line of slope $-1$ from $(0,1)$

$$\begin{aligned}
4 * t^2 - 4t + 1 &= 1 - t \\
4 * t^2 - 3t &= 0 \\
t(4t - 3) &= 0
\end{aligned}$$

The second factor give us the intersection after 0 we are interested in

$$t_{max} = 3/4 = 0.75$$

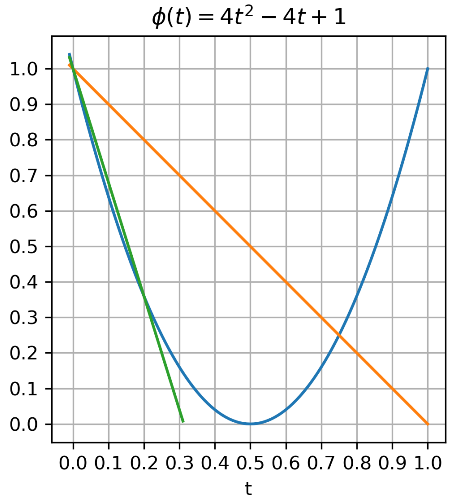



Figure 1: lines: orange, slope damped by $\alpha = 0.25$ and green by $\beta = 0.8$

The lower limit can be obtained by solving for the slope $\phi'(t)$ equal $\beta\phi'(0)$

$$8t - 4 = -3.2$$
$$t_{min} = (4 - 3.2)/8 = 0.8/8 = 0.1$$

## 6 Active Set Method

**6.1**. Consider the following 2-dimensional minimization problem

$$\begin{array}{llr} \text{min} & q(x,y) = (x-1)^2 + (y-2.5)^2 & \\ \text{subject to:} & x - 2y + 2 \geq 0 & (1) \\ & -x - 2y + 6 \geq 0 & (2) \\ & -x + 2y + 2 \geq 0 & (3) \\ & x \geq 0 & (4) \\ & y \geq 0 & (5) \end{array}$$

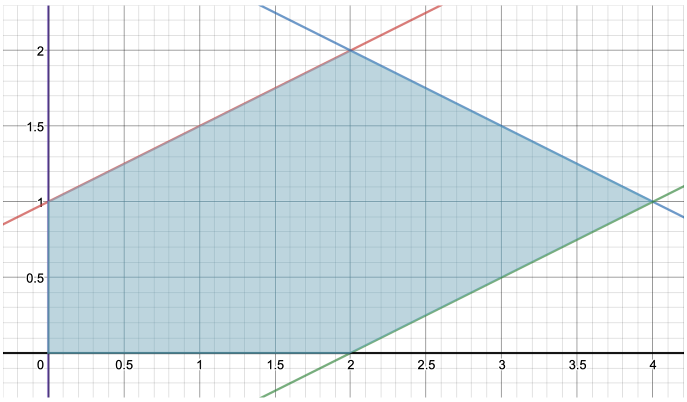
(a) Sketch the feasible set.

(b) Assume we start from point $(1, 1/2)^T$, what are the active constraints?

(c) Start from the initial state of (b), write down the steps of the active set method until it reaches the optimum point. In each step, show the KKT system, active set and solution.

*Solution:*

(a) The feasible set is sketched in the following figure.




(b) The active constraint set is empty.

(c) Start point $\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$. Step1: $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$. The solution of the system is $(1, 2.5)^T$. With the full step, the point will be outside the feasible region. The blocking constraint is 1 and the intersection is at $(1, 1.5)^T$. $\begin{pmatrix} 1 \\ 0.5 \end{pmatrix} + \alpha \begin{pmatrix} 1-1 \\ 2.5-0.5 \end{pmatrix} = \begin{pmatrix} 1 \\ 1.5 \end{pmatrix}$. So with step length 0.5, we get the solution $(1, 1.5)^T$. The active set is $\{1\}$.

Step2: $\begin{pmatrix} 2 & 0 & -1 \\ 0 & 2 & 2 \\ -1 & 2 & 0 \end{pmatrix}\begin{pmatrix} x_2 \\ y_2 \\ \lambda_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \\ 2 \end{pmatrix}$. The solution is $(1.4, 1.7, 0.8)^T$. The active set is $\{1\}$.

Step3: $\begin{pmatrix} 2 & 0 & -1 \\ 0 & 2 & 2 \\ -1 & 2 & 0 \end{pmatrix}\begin{pmatrix} x_3 \\ y_3 \\ \lambda_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \\ 2 \end{pmatrix}$. The solution is $(1.4, 1.7, 0.8)^T$ as last step. We can see that the $\lambda_1$ for constraint 1 is larger than 0. So the algorithm terminates.

# 7 Algorithms

**7.1**. Newton's Methods

(a) Why does Newton's method require that the Hessian matrix is positive definite?

*Solution:* We know that $H\Delta x = -g$, where $H$ is the hessian matrix and $g$ is the gradient. To guarantee the line search is able to find a descent direction, we ask for $g^T \Delta x < 0$. This results in $g^T \Delta x = -g^T H^{-1} g < 0$, which means $H$ should be positive definite.

(b) In the projected Newton's method, what do we do to the hessian matrix so that the algorithm applies for non-convex problems?

*Solution:* If the Hessian is not positive definite, we modify the eigenvalues so that they are all positive. One way is to add a multiple of identity matrix to shift the eigenvalues until they all get positive.

# 8 Programming

**8.1.** The following is the implementation of Newton's method. Indicate the wrong code and give the correction.

```
// get starting point
int n = _problem->n_unknowns();
Vec x(n);
_problem->initial_x(x);
// allocate gradient storage
Vec g(n);
// allocate hessian storage
SMat H(n, n);
// allocate search direction vector storage
Vec delta_x(n);
int iter(0);

Eigen::SimplicialLDLT<SMat> solver;
do {
    ++iter;

    // solve for search direction
    _problem->eval_gradient(x, g);
    _problem->eval_hessian(x, H);

    // solve for delta_x
    solver.compute(H);
    delta_x = solver.solve(g);    // Error(1): the lhs should be -g

    // Newton decrement
    double lambda2 = g * delta_x; // Error(2): should use dot product

    // Stop check
    if (lambda2 <= eps_) break;

    // step size
    double t = LineSearch::backtracking_line_search(...);

    // update
    x += t * delta_x;
} while (iter < _max_iters);
```

Solution:

(1) `delta_x = solver.solve(-g);`

(2) `double lambda2 = -g.transpose()*delta_x;`