Advanced Topics in Operations Research Notes

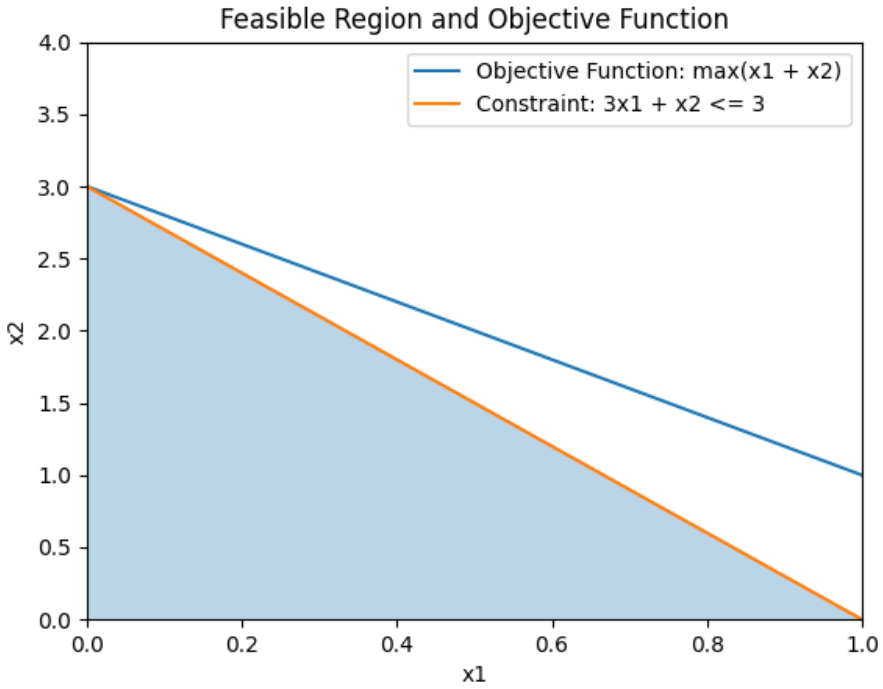
Andres Espinosa

September 16, 2024

Contents

1 Wednesday 08/28/2024

$$max\ x_1 + x_2 \tag{1}$$
$$s.t.\ 3x_1 + x_2 \leq 3 \tag{2}$$
$$x_1 \geq 0, x_2 \geq 0 \tag{3}$$



x^* is an extreme point if we cannot find $x_1, x_2 \in S$, such that $x_1 \neq x_2$ and $x^* = \lambda x_1 + (1-\lambda)x_2$ for some $\lambda \in (0, 1)$
For an n-dimensional problem, there are n-many constraints that are active/binding and their coefficients are linearly independent.
Optimal solutions can be found that are not basic feasible solutions when constraints and objective functions lie on the same plane.

2 Friday 08/30/2024

2.1 Complexity Introduction

Three elements for complexity analysis:

- Model or problem formulation - The known part of a problem. Includes the formulation and problem descriptions.
- Oracle - The smallest computing unit, whose details inside can be ignored.
- Target outcome - When to stop

Two types of complexity:

- Analytical complexity - The number of calls of the oracle which is necessary to solve a given problem formulation up to accuracy ϵ
- Numerical complexity - The number of arithmetic operations which is necessary to solve a given problem formulation up to accracy ϵ

2.2 P, NP, NP-hard

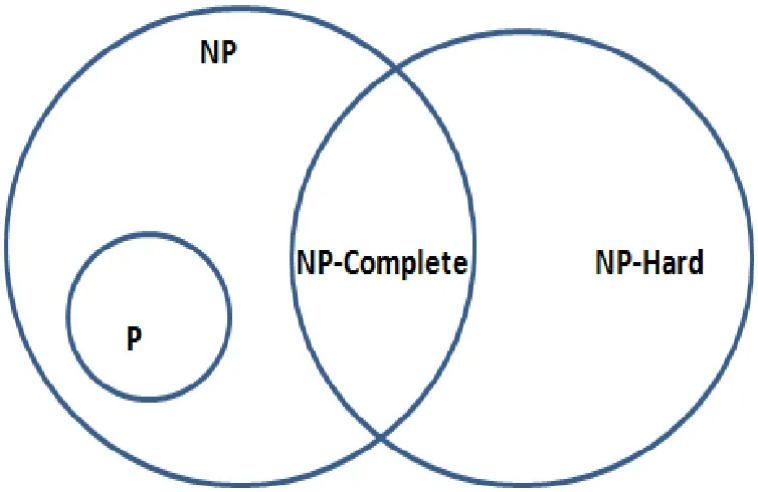
A **problem** is a function $F : I \rightarrow B$, where I is the set of instances encoded as strings of characters and B is the set of problem outputs.

For example, linear programming in $\mathbb{R}^{mxn}$ is a problem where the instance parameters are (m, n, A, c, b) and the output B is the optimal solution found to the objective function.
Another example, making a decision takes in an instance I and outputs a decision of yes or no.

$$I = \textit{Problem-specific information parameters}, B \in (yes, no) \tag{4}$$

3 Wednesday 09/04/2024

3.1 P, NP, NP-complete, and NP-hard



A **problem** is a function $F : I \to B$, where I is the set of instances encoded as strings of characters and B is the set of problem outputs.

A mapping from I to B by taking in the input parameters of what the specific problem actually entails, and then the solution or answer is parametrized by B.

For example, linear programming in $\mathbb{R}^{mxn}$ is a problem where the instance parameters are (m, n, A, c, b) and the output B is the optimal solution found to the objective function.

Another example, making a decision takes in an instance I and outputs a decision of yes or no.

$$I = \textit{Problem-specific information parameters}, B \in (yes, no) \tag{5}$$

3.1.1 Turing machine

A turing machine is a model of how a computer processes a program. There is a *tape* and *finite-state controller*

- **Tape**: A memory unit divided into tape cells. Each tape cell stores one symbol chosen from a finite set of symbols. The tape head is the sole cell currently processed by the controller.
- **Finite-state controller**: A collection of finite states and rules where at each time step, the controller takes an action of erase, move, update cell, and update state.

There is only one unique rule applied for each symbol-state combination

A turing machine is said to be able to solve a problem $F : I \to B$, if given any instance $x \in I$, the turing machine eventually produces the solution B and stops.

My understanding of this is that a turing machine has a controller that goes and selects individual cells based off of the symbol and then makes an action to change the state based off of a set of symbol-state pairs.

3.1.2 P-problem

The class P of problems is defined to be those problems $F : I \to yes, no$ such that a Turing machine M can compute F, and the number of steps required by M for this computation is bounded by $p(n)$ where p is a polynomial and n is the length of the input.

In other words, a problem P can be solved in polynomial time by a turing machine.

F is a decision problem

NP problem: Problems whose solution, if it is "yes", have proofs verifiable in polynomial time by a deterministic Turing machine.

P problems can be solved in polynomial time, NP problems are problems that can be verified in polynomial time. A non-deterministic Turing machine would be able to solve an NP problem in polynomial time. *A non-deterministic Turing machine is defined as the relaxation of the "unique rule per symbol-state" constraint.*
For example, in the exercise done in class, instead of having the unique rule of "if no one is to the right, say 'yes'", you could have a non-deterministic rule where you have a 50% chance of saying 'no' or 'yes'. Putting randomness in the rule would make it a non-deterministic machine.

3.1.3 NP-complete

A decision problem $F : I \to yes, no$ can be polynomial transformed into another decision problem $G : I' \to yes, no$ if there exists a function $f : I \to I'$. The function f can be computed by a polynomial-time Turing machine, and f has the property that $F(x) = yes \iff G(f(x)) = yes$
Suppose $F : I \to yes, no$ is a decision problem that lies in NP, and suppose that every problem in NP can be polynomially transformed to F . Then, F is said to be NP-complete

An algorithm can solve all NP problem efficiently if it can solve an NP-complete problem efficiently. If it solves an NP-complete problem, that means it is able to transform a problem into an NP problem.

4 Friday 09/06/2024

I did not attend this day so just notes over the slides.

4.1 NP-hard

4.1.1 Definitions

We say that a Turing machine can use F as an **oracle** if M is augmented with the following capability: M can compute $F(x)$ from x in one step.
We say that a problem G is **Cook-reducible** to problem F (aka, polynomial-time Turing reducible) if G can be computed in polynomial time by a Turing machine that uses F as an oracle.
A problem F is said to be **NP-hard** if an NP-complete problem is Cook-reducible to F.

4.1.2 Quadratic problem

General quadratic programming:

- Given symmetric matrix $H \in \mathbb{R}^{nxn}$, matrix $A \in \mathbb{R}^{mxn}$, and vectors **b,c** $\in \mathbb{R}^n$
- Solve

$$\begin{aligned} \min \mathbf{x}^T H\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ s.t. A\mathbf{x} \geq \mathbf{b} \end{aligned} \tag{GQP}$$

5 Monday 09/09/2024

I also did not attend today so these are slide notes

5.1 Nonlinear Programming Basic Concepts

5.1.1 Nonlinear problem formulation

$$\begin{aligned} &\min_x f(\mathbf{x}) \\ \text{s.t. } &h(\mathbf{x}) = 0 \\ &g(\mathbf{x}) \leq \mathbf{0} \end{aligned} \tag{6}$$

where $f : \mathbb{R}^n \to \mathbb{R}, h : \mathbb{R}^n \to \mathbb{R}^{m_1},$ and $g : \mathbb{R}^n \to \mathbb{R}^{m_2}$

- Objective function, cost function, disutility function
- Inequality constraints, non-negativity constraints
- Equality constraints
- Feasible region, feasible solution
- Optimal solution, minimal solution
- Unconstrained problems

5.1.2 Global and local solutions definitions

Let $\Omega := \mathbf{x} : g(\mathbf{x} \leq \mathbf{0}, h(\mathbf{x} = \mathbf{0}))$

A feasible solution $\mathbf{x}^* \in \Omega$ is said to be globally minimal if and only if $f(\mathbf{x}^*) \leq f(\mathbf{x}) \forall \mathbf{x} \in \Omega$

A feasible solution $\mathbf{x}^* \in \Omega$ is said to be locally minimal if and only if $f(\mathbf{x}^*) \leq f(\mathbf{x}) \forall \mathbf{x} \in \Omega \cap N(\mathbf{x}^*, \epsilon)$ for some $\epsilon > 0$. Here, $N(\mathbf{x}^*, \epsilon) = \{\mathbf{x} : ||\mathbf{x}^* - \mathbf{x} \leq \epsilon||\}$

In other words, a global solution is global if it is the minimum of all values in the set. It is locally minimal if it is less than all points in a certain radius defined arbitarily by ϵ.

5.2 Necessary conditions of optimality

5.2.1 Gradients

Consider a function $f\mathbb{R}^n \to \mathbb{R}$, then the gradient of f is given as

$$\nabla f(\mathbf{x}) := \begin{bmatrix} \frac{df(\mathbf{x})}{dx_1} \\ \frac{df(\mathbf{x})}{dx_2} \\ \vdots \\ \frac{df(\mathbf{x})}{dx_n} \end{bmatrix} \tag{7}$$

Finding gradient of function $f(x_1, x_2) = 0.5x_1^2 + 0.5x_2^2$ at (1,-1)

$$\frac{df(\mathbf{x})}{dx_1} = x_1 \tag{8}$$

$$\frac{df(\mathbf{x})}{dx_2} = x_2 \tag{9}$$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{10}$$

5.2.2 Examples

a.

$$\nabla 2x_1^2 + 2x_2^2 = \begin{bmatrix} 4x_1 \\ 4x_2 \end{bmatrix} \tag{11}$$

Solving for $\nabla f = 0$,

$$\mathbf{x}^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{12}$$

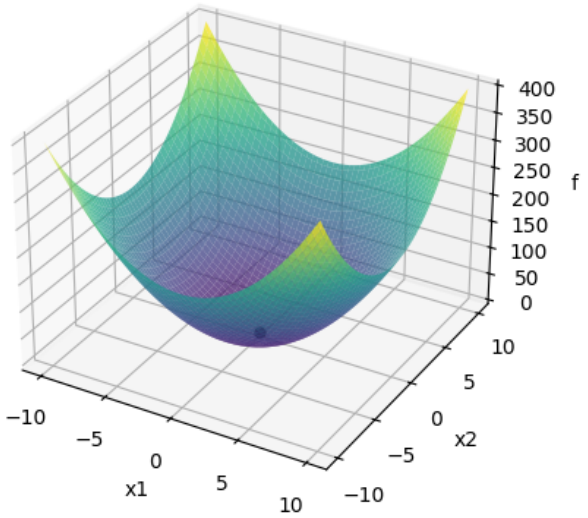


Figure 1: Graph of $2x_1^2 + 2x_1^2$ with solution at $(x_1, x_2) = (0, 0)$

b.

$$\frac{d\sin(x)}{dx} = \cos(x) \tag{13}$$

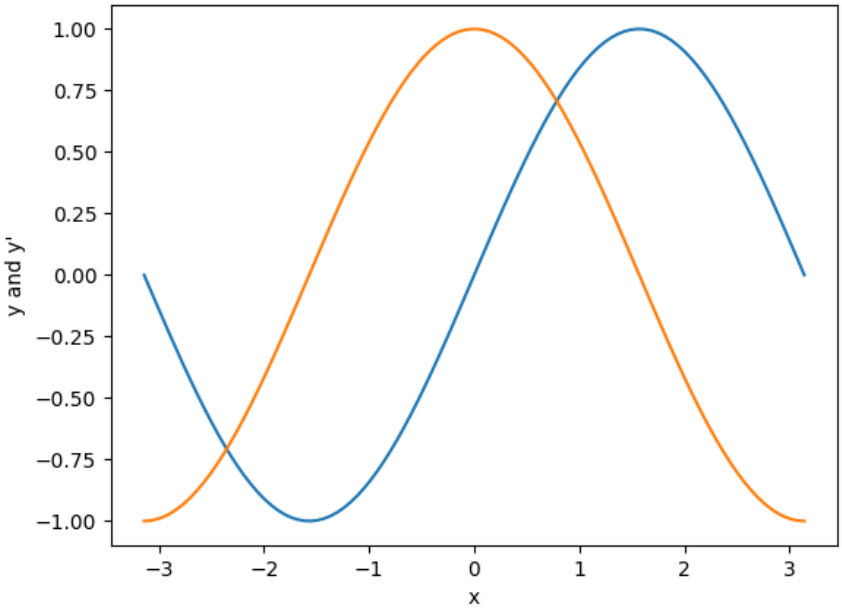


Figure 2: Graph of sin(x) and the solution cos(x)

Solution to this problem is the curve $\cos(x)$
c.

$$\frac{dx^3}{dx} = 3x^2 \tag{14}$$

Solution to this problem is the curve $3x^2$
d. $(x-3)(x^2-1)(x+2)$ is already in it's factored form so with product rule:

$$\nabla f = 4x^3 - 3x^2 - 14x + 1 \tag{15}$$

- x = -1.5
- x = 2
- x = 0

6 Wednesday 09/11/2024

6.1 Nonlinear programming Part II

6.1.1 Problem formulation

$$\begin{aligned} &\min_x f(x) \\ &s.t. g(x) \leq 0 \end{aligned} \tag{16}$$

Now there is a constraint on the problem

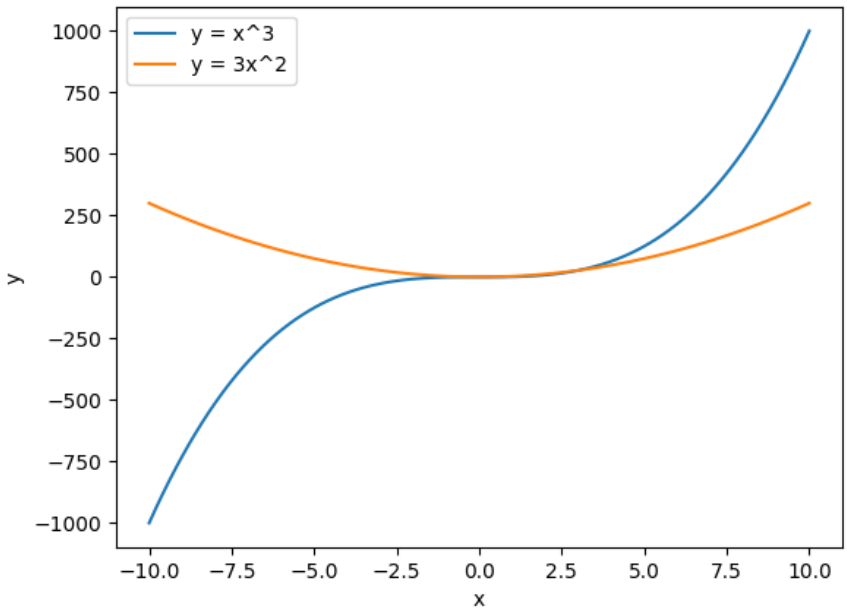


Figure 3: Graph of x^3 and it's solution $3x^2$

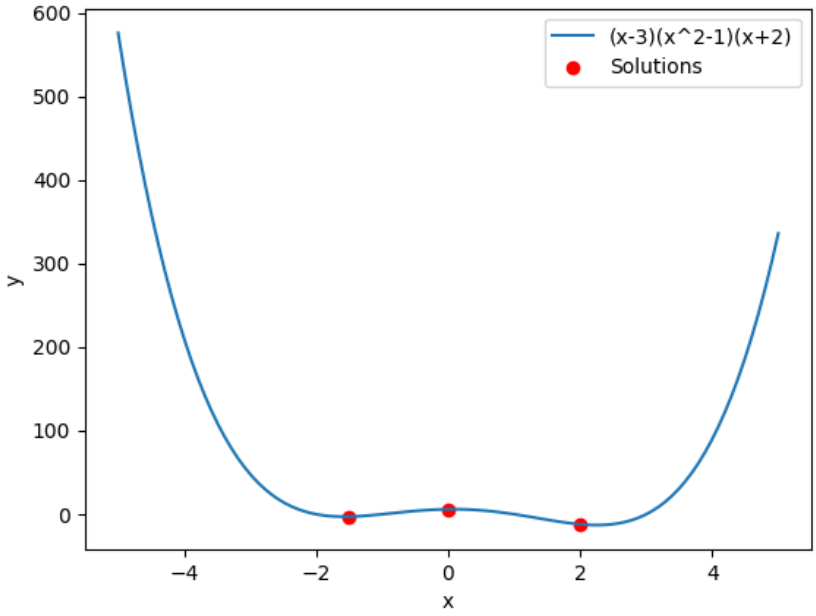


Figure 4: Graph of $(x-3)(x^2-1)(x+2)$ and it's solutions $x = 0, -1.5, 2$

6.1.2 Gradient descent intro

Dropping a coin into a bowl will cause the coin to travel in the direction of the negative gradient until it reaches $\nabla f = 0$

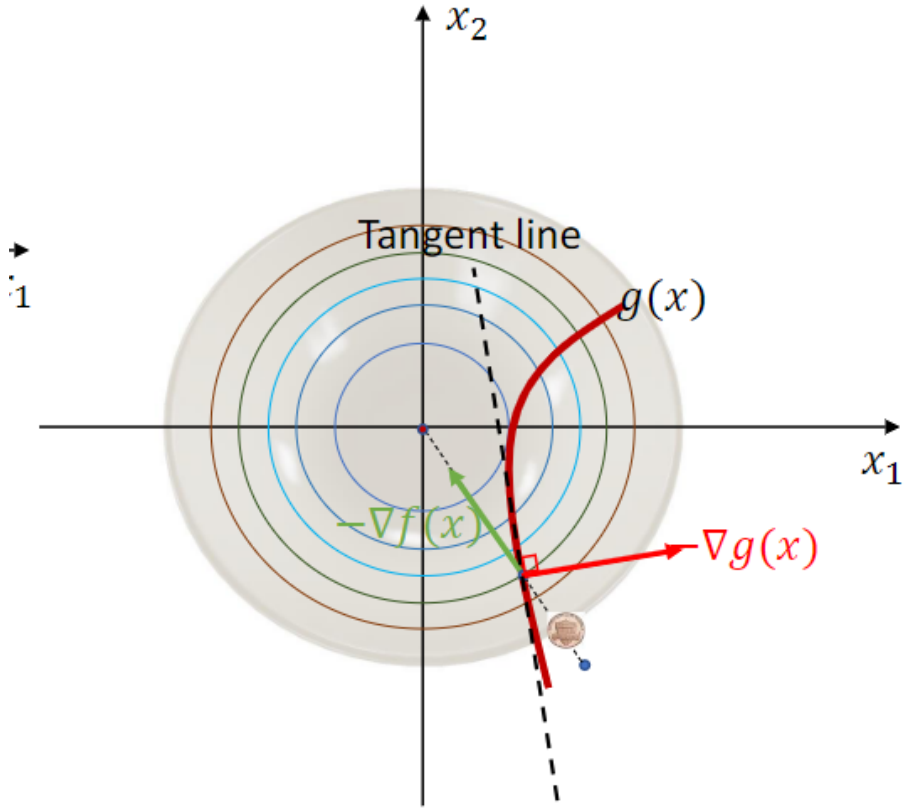


Figure 5: A coin traveling in the direction of $-\nabla$ to reach the bottom of the bowl

$$f(x + \delta d) - f(x) = \langle \nabla f(x), \delta d \rangle + 0(\delta^2) = \langle \nabla f(x), \delta d \rangle \tag{17}$$

In the event that $\langle \nabla f(x), \delta d \rangle < 0$, The step is in a direction that is away from the gradient. It also implies that the gradient is not zero at this point.

We can also introduce a constraint function $g(x)$ present in Figure 5 that defines the section (or in the figure's case, a line) that the function f is feasible on. With the introduction of this constraint, the coin can only travel along the path of the constraint g. For the solution to be optimal in the context of the function f subject to the constraint g, we would need to reach the point where $\nabla f(x^*) + \gamma \nabla g(x^*) = 0$, where $\gamma = \frac{\|\nabla f\|}{\|\nabla g\|}$

7 Friday 09/13/2024

7.1 Review

7.1.1 What is an Inner Product

An inner product defined as:

$$\langle u, v \rangle \tag{18}$$

is a generalization of the dot product, it effectively gives the answer to the question: How much of one vector is in the direction of the other vector?

7.1.2 Understanding $g(x^*) = 0$

If $g(x^*) \leq 0$, then the constraint is being obeyed. The constraint is not obeyed otherwise.

Taking the equation,

$$g(x^* + \delta d) = \delta \langle \nabla g(x^*), d \rangle - \epsilon \tag{19}$$

The step size δcan be moved, and as long as that term is less than zero and does not breach the feasible region.

This implies that for a solution to be optimal, $g(x^*) = 0$ must be true, (the converse does not hold).

7.2 Gradient bowl

7.2.1 Adding another constraint

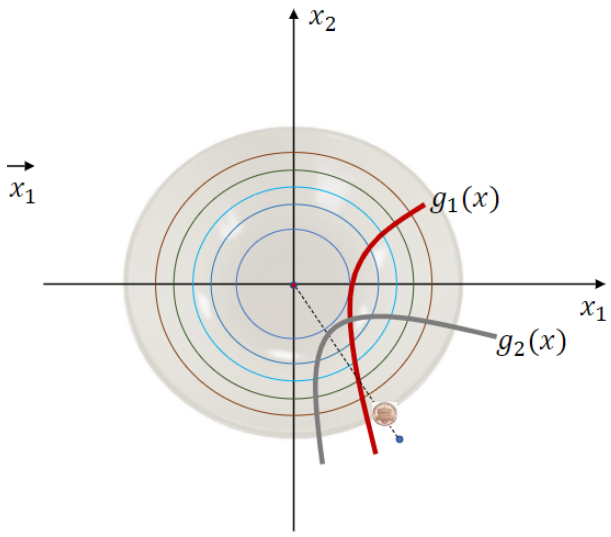


Figure 6: Image of the bowl with a second constraint

We want to walk opposite of the direction of ∇f while still walking perpendicular to the gradient

of the constraint.

$$\begin{aligned} &\min f(x) \\ s.t. &g_1(x) \leq 0 \\ &g_2(x) \leq 0 \end{aligned} \tag{20}$$

A cone is a collection of rays that start from a certain point.

In order to find the set of directions we can walk along that will be both improving the objective function while staying in the feasible region, we need to find the intersection of the halfspaces created by each.

- The feasible region based off of g_1 is the halfspace created by the set $\{x|\nabla g_1(x) \leq 0\}$.
- The feasible region based off of g_2 is the halfspace created by the set $\{x|\nabla g_2(x) \leq 0\}$.
- The "improving" region based off of f is the halfspace created by the set $\{x|\nabla f(x) \leq 0\}$.

The intersection of these sets is the cone of all possible directions that would result in an improvement to the function while obeying the constraints.

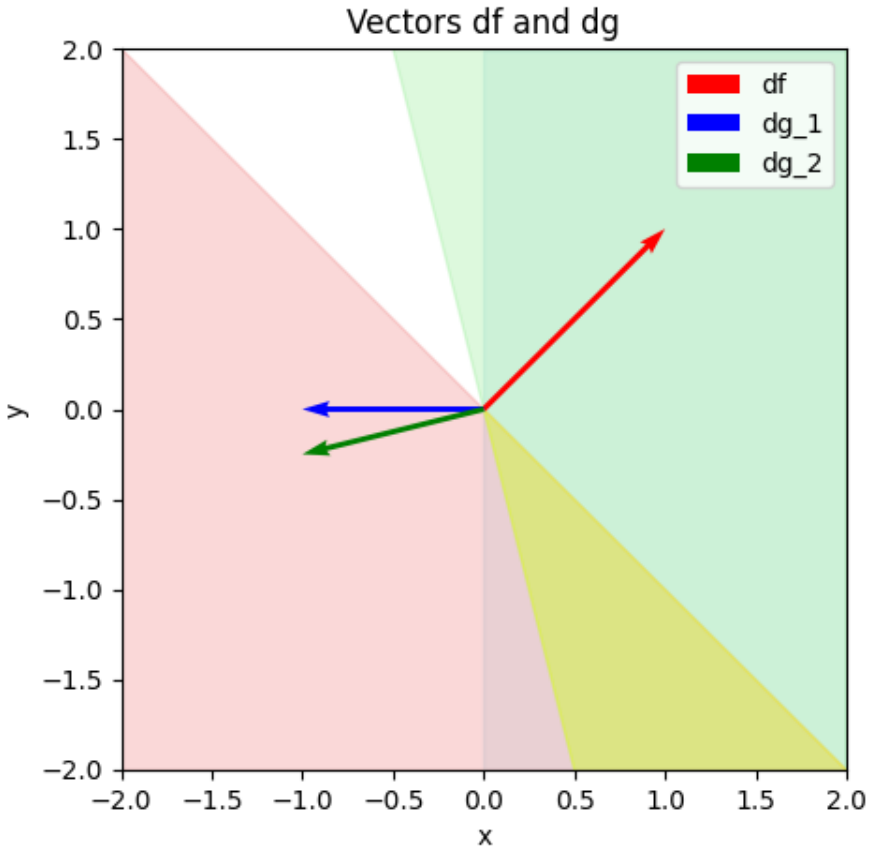


Figure 7: Depicting the walkable region as the intersection between the three regions

7.2.2 Finding optimality conditions

For two constraints g_1 and g_2, the solution x^* is optimal if

$$\nabla f(x^*) + \gamma_1 \nabla g_1(x^*) + \gamma_2 \nabla g_2(x^*) = 0 \tag{21}$$

8 Monday 09/16/2024

8.1 Review

8.1.1 Problem formulation for multiple constraints

$$\begin{aligned} \min f(x) \\ s.t. g_1(x) \leq 0 \\ g_2(x) \leq 0 \end{aligned} \tag{22}$$

The difference between a function and a tiny step of size δ in direction d can be approximated as the inner product between the gradient of the function and the step δd

$$\begin{aligned} f(x+\delta d) - f(x) &\approx \langle \nabla f(x), \delta d \rangle \\ g_1(x+\delta d) - g_1(x) &\approx \langle \nabla g_1(x), \delta d \rangle \\ g_2(x+\delta d) - g_2(x) &\approx \langle \nabla g_2(x), \delta d \rangle \\ g_1(x) = 0, g_2(x) &= 0 \end{aligned} \tag{23}$$

In order for the constraints set in the general minimization problem to be met, the inner product between the gradient of the constraint and the step should be non-positive. We can define the "walkable" region as the intersection of the halfspaces created by the negative gradients of the objective function and any constraints. For a solution to be optimal, this intersection must be empty.

8.1.2 Optimality

A minimal solution is reached when

$$\nabla f(x^*) + \gamma_1 \nabla g_1(x^*) + \gamma_2 \nabla g_2(x^*) = 0 \tag{24}$$

where $\gamma_1 \geq 0, \gamma_2 \geq 0$

This is a non-negative linear combination of the gradient vectors. The set of all non-negative linear combinations of two vectors u and v is a cone with edges on u and v. This cone can be parametrized by a single scalar w using the following

$$\begin{aligned} w(r_1 u + r_2 v) &= y \\ r_1 + r_2 &= 1 \\ r_1 \geq 0, r_2 \geq 0, w &\geq 0 \end{aligned} \tag{25}$$

Using that logic to explain the equation $\nabla f(x^*) + \gamma_1 \nabla g_1(x^*) + \gamma_2 \nabla g_2(x^*) = 0$, there is a vector ∇f that is opposite of any linear combination of the constraints.

8.2 Active constraints

In an optimization problem such as the bowl, it is possible that there is only one active constraint, to the problem. Therefore, although g_1 and g_2 are defined, only one is relevant to the optimization problem.

$$\nabla f(x^*) + \gamma_1 \nabla g_1(x^*) + \gamma_2 \nabla g_2(x^*) = 0 \tag{26}$$

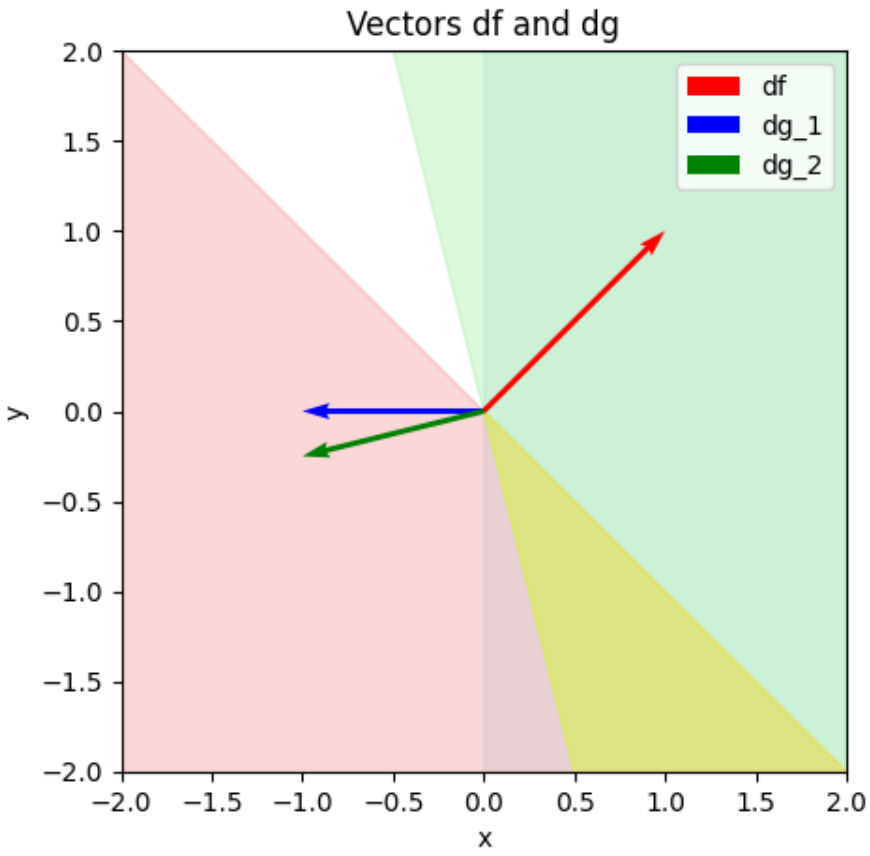


Figure 8: Image of an empty walkable region since ∇f is opposite of ∇g

where $\gamma_1 \geq 0, \gamma_2 \geq 0$ when there is only one active constraint, is equivalent to

$$\nabla f(x^*) + \gamma_1 \nabla g_1(x^*) = 0 \tag{27}$$

where $\gamma_1 \geq 0$

Can we alter the first rule to incorporate this new information of potentially inactive constraints? Adding binary variables.