Basic Optimization Theory

What is a Optimization Algorithm

Optimization algorithms are search methods where the goal is

- to find a optimal solution to an optimization problem
- or find the "good enough" solution, when finding the optimal solution is infeasible.

There are often additional constraints to consider as well.

What is a optimization algorithm

The study of optimization algorithms is very prevalent in the following fields

- Mathematics
- Operational Research
- Computer Science
- Statistics

But the application of optimization techniques spans almost every field.

The Optimization Problem

Optimization problem generally consists of the following basic ingredients

- An **objective function**, which represents the quantity to be optimized, that is, the quantity to be minimized or maximized.
- A set of **unknowns** or **variables**, which affects the value of the objective function
- A set of **constraints**, which restricts the values that can be assigned to the unknowns.

The Optimization Problem

Unconstrained minimization optimization problem in $\mathbb{R}^n$

Given an objective function $f : \mathbb{R}^n \to \mathbb{R}$, find a $\boldsymbol{x}^* \in \mathbb{R}^n$ such that

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \mathbb{R}^n \tag{1}$$

Domain/Boundary only constrained minimization optimization problem $\mathbb{R}^{n*}$

Given an objective function $f : \boldsymbol{D} \to \mathbb{R}$, find a $\boldsymbol{x}^* \in \boldsymbol{D} \subset \mathbb{R}^n$ such that

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \boldsymbol{D} \tag{2}$$

*problems that are only domain constrained are often refereed to as unconstrained problems.

Optima types

Generally is not possible find a true global optima. Let X be the search space

Global Minimum

$\boldsymbol{x}^*$ is a global optima if

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in X \tag{3}$$

Optima Types

Strong Local Minimum

$\boldsymbol{x}^* \in S \subset X$ is a strong local minimum if

$$f(\boldsymbol{x}^*) < f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in S/\{\boldsymbol{x}^*\}, \tag{4}$$

and S is convex.

Weak Local Minimum

$\boldsymbol{x}^* \in S \subset X$ is a weak local minimum if

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in S/\{\boldsymbol{x}^*\}, \tag{5}$$

and S is convex.

Convexity

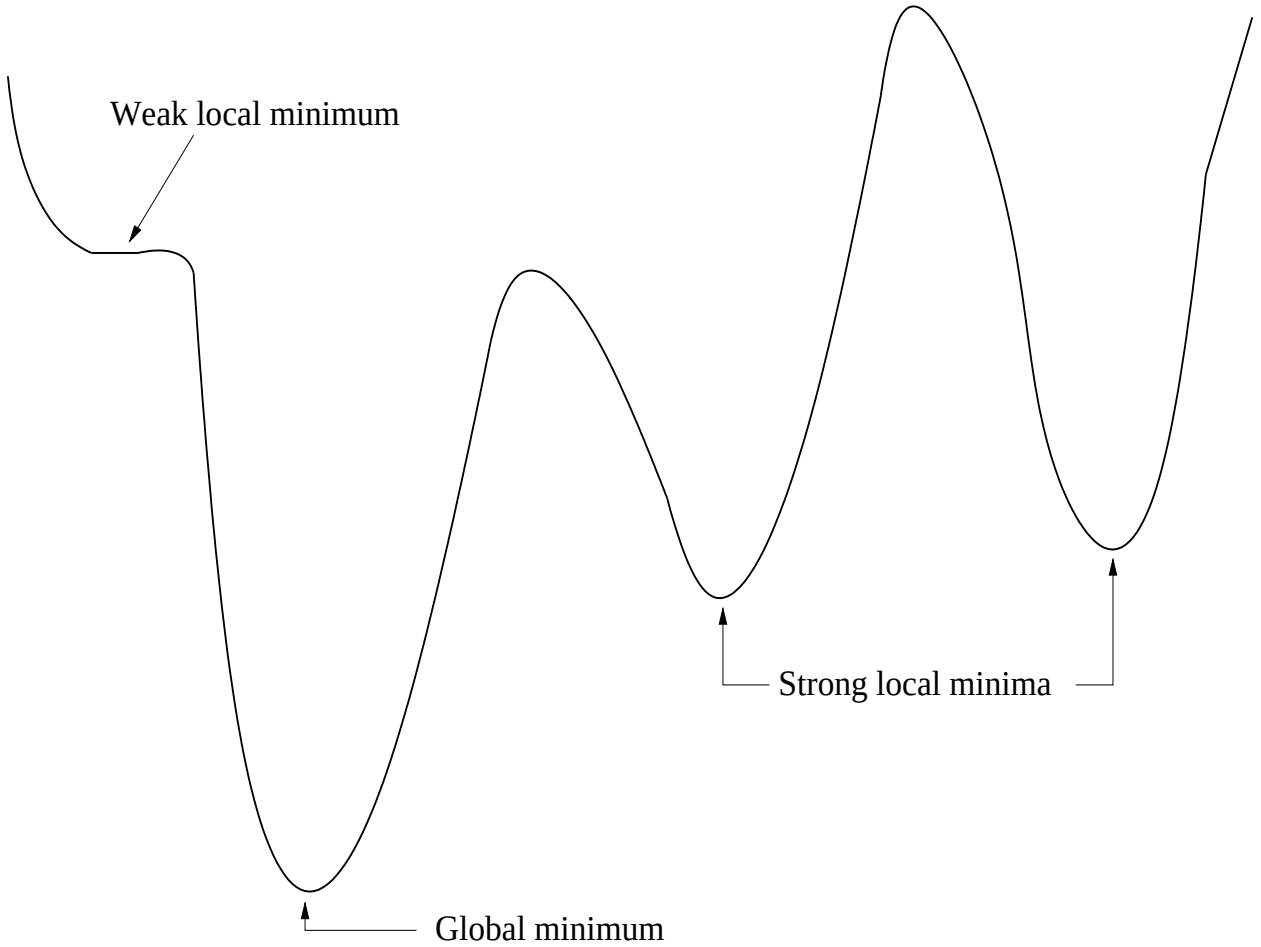
Convex set

- Let $\mathbf{x}, \mathbf{y} \in A \subset X$, a vector space.
- A is convex if $\mathbf{f}(t) \in A$ for $0 \leq t \leq 1$ where $\mathbf{f}$ is defined as

$$\mathbf{f}(t) = t\mathbf{x} + (1-t)\mathbf{y} \tag{6}$$

- Or said differently $Range(\mathbf{f}(D)) \subseteq A$, where $D = [0, 1]$

Optima Types



Optimization Method Classes

The are two broad classes

- Deterministic optimization.
- Stochastic optimization.

Optimization Method Classes

And many subclasses

- Unconstrained methods
- Constrained methods
- Discrete(combinatorial), continuous, or mixed domain method
- Multi-objective optimization methods
- Multi-solution (niching) methods
- Dynamic methods

These methods can be either single-point or multi-point based.

General Local Search Procedure

Local search methods generally follow a certain basic structure

General Local Search Algorithm

Find starting point $\mathbf{x}(0) \in \mathcal{S}$;
$t = 0$;
repeat
    Evaluate $f(\mathbf{x}(t))$;
    Calculate a search direction, $\mathbf{q}(t)$;
    Calculate step length $\eta(t)$;
    Set $\mathbf{x}(t+1)$ to $\mathbf{x}(t) + \eta(t)\mathbf{q}(t)$;
    $t = t + 1$;
until *stopping condition is true*;
Return $\mathbf{x}(t)$ as the solution or $\max_t \mathbf{x}(t)$ if you track the path;

General Local Search Procedure

There are numerous ways of calculating the direction and magnitude of movement.

- Gradient Decent/Steepest descent
- Conjugate Gradient Methods (very low number of needed steps)
- Quasi-Newton Methods
- Trust Region Methods
  - Popular in model-free reinforcement learning.

Gradient Decent (GD)

Most common in machine learning

- Let $E(\mathbf{W}, \mathbf{i}, \mathbf{t})$ be the error of a model parameterized by the weight matrix $\mathbf{W}$. $\mathbf{i}$, and $\mathbf{t}$ are the input and targets respectively.
- Gradient decent simply moves along the negative partial derivative of $E(\mathbf{W}, \mathbf{i}, \mathbf{t})$ for each weight
  - $w_{i,j}(t+1) = w_{i,j}(t+1) - \eta \frac{\partial E}{w_{i,j}}$
  - $\frac{\partial E}{w_{i,j}}$ is the search direction.
  - η is the step length.

Beam Search

Some approaches are better suited to discrete problem spaces, such as Beam Search (BS). BS is a classical tree local search method.

- Beam search uses breadth-first search to build its search tree.
- At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost.
- However, it only stores a predetermined number, b_w beam width, of best states at each level.

In essence it is a breadth first search with a greedy policy for deciding which paths to ignore.

- It is very memory efficient, for a breadth first type search.

Tabu Search

Tabu search (TS) is another search algorithm which is often used for solving combinatorial optimization problems.

- The abstract premise is that you mark certain candidate solution as **tabu**, as a way to prevent an iterative local search algorithm from continually revisiting solutions.
- The method helps avoid, and escape local optima.

Consider a greedy hill climber on a grid.

- Normally you would move to the position with the best objective function value.
- With TS you move to the best non-"tabu"ed position.

Tabu Search: Simple Grid Search

Consider the problem of searching for the global minimum.

1	2	3	4	6
1	2	1.5	4	6
2	2	3	3	6
2	2	5	4	5

- Possible issue with vanilla hill-climber?
- Would a Tabu visit threshold help?

Simulated Annealing

Simulated annealing is an stochastic optimization algorithm process based on the physical process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system energy.

- The simple ideas is that both good and poor candidate solutions will be visited/accepted but the likelihood of visiting a poor solution decreases over time.
- The likelihood is also based on the degree by which the new solution would decrease the optimality by.
- Probability of selection is derived from a Boltzmann Gibbs distribution.

Simulated Anneling

More specifically SA, have the following components

- Representation of possible solutions
- Mechanism to generate new solutions
- Method to evaluate solutions,
- Annealing schedule

Simulated Annealing

Simple way of generating a new solution.

- If $\mathbf{x}(t)$ is the current solution.
- Generate a **possible** new solution using

$$\mathbf{x}(t) + \mathbf{D}(t)\mathbf{r}(t) \tag{7}$$

where $\mathbf{r}(t) \sim U(-1, 1)^d$ and $\mathbf{D}(t)$ is a diagonal matrix that defines the maximum change allowed in each variable.

- in it's simplest form $\mathbf{D}(t) = \mathbf{D}$.
- But often

$$\mathbf{D}(t+1) = (1-\alpha)\mathbf{D}(t) + \alpha\omega\mathbf{R}(t) \tag{8}$$

where $\mathbf{R}(t)$ is a diagonal matrix whose elements are the magnitudes of the successful changes made to each variable, and α and ω are constants.

Simulated Annealing

Two common and simple annealing Schedules are

- Exponential cooling: $T(t+1) = \alpha T(t)$.
- Linear cooling: $T(t+1) = T(t) - \Delta T$, where $\Delta T = \frac{T(0) - T(FinalIterCount)}{FinalIterCount}$

In general, these are used as cooling schedules, so

- Start high
- End Low

Simulated Annealing

Create initial solution, $\mathbf{x}(0)$ and set initial temperature, $T(0)$, $t = 0$;

repeat

Generate new solution, $\mathbf{x}$, and determine the quality, $f(\mathbf{x})$;

Calculate acceptance probability using

$$P = \begin{cases} 1 & \text{if } f(\mathbf{x}) < f(\mathbf{x}(t)) \\ e^{-\frac{f(\mathbf{x})-f(\mathbf{x}(t))}{c_b T}} & \text{otherwise} \end{cases}$$

if $U(0,1) \leq P$ **then**

$\mathbf{x}(t+1) = \mathbf{x}$;

end

t=t+1;

until *stopping condition is true*;

Return $\mathbf{x}(t)$ as the solution;

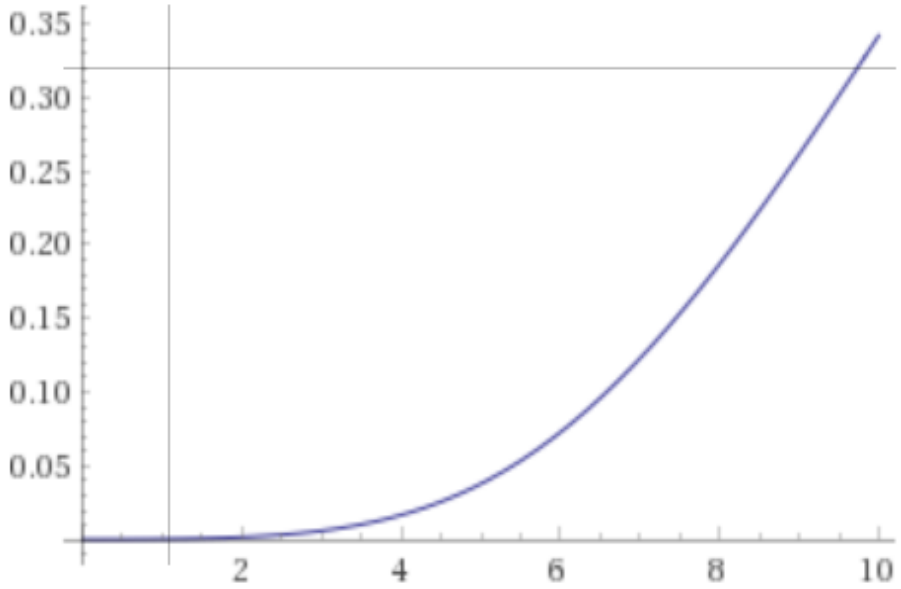
* note c_b the Boltzmann constant

Simulated Annealing

Input interpretation:

plot	$e^{-10/1.25^T}$	$T = 0$ to 10

Plot:



Test Problems

In order to evaluate the effectiveness of optimization algorithms, we need a suite of test functions with which to compare performance. Two of which I will show here are

- Ackley
- Rastrigin

However nice collections can be found in:

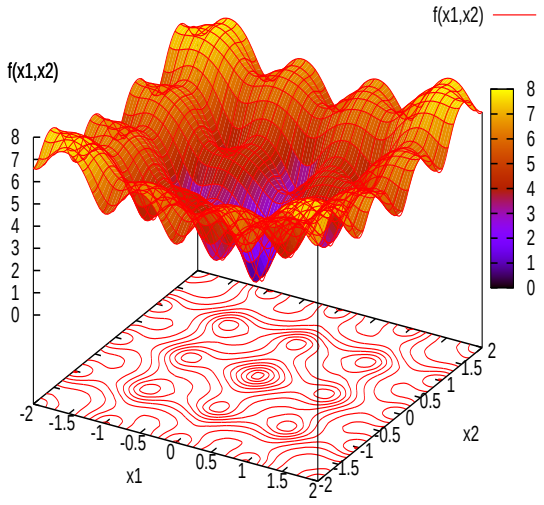
- K. Malan, Characterising continuous optimisation problems for particle swarm optimisation performance prediction, PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2014.
- A. Engelbrecht, Particle swarm optimization: Global best or local best, in Proceedings of the 1st BRICS Countries Congress on Computational Intelligence. Piscataway, NJ: IEEE Press, 2013, pp. 124135.
- J. Liang, B. Qu, and P. Suganthan, Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization, Computational Intelligence Laboratory, Zhengzhou University and Nanyang Technological University, Tech. Rep. 201311, 2013.

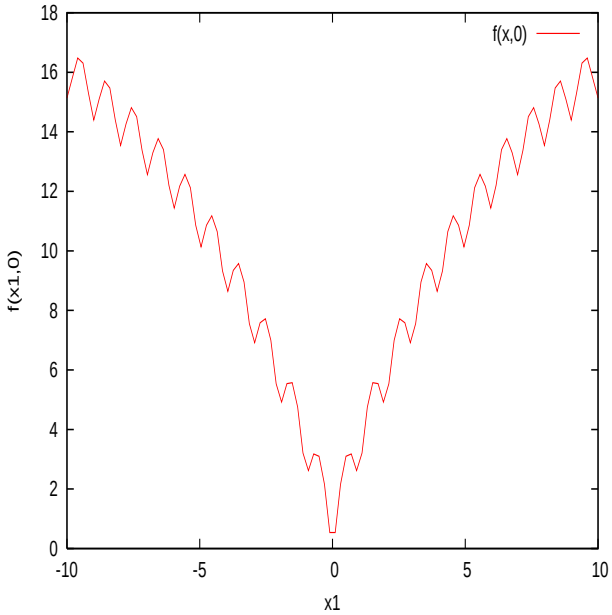
The Ackley Problem

Definition:

$$f(\mathbf{x}) = -20e^{-0.2\sqrt{\frac{1}{n_x}\sum_{j=1}^{n_x} x_j^2}} - e^{\frac{1}{n_x}\sum_{j=1}^{n_x}\cos(2\pi x_j)} + 20 + e \qquad (9)$$

with $x_j \in [-30, 30]$ and $f^*(\mathbf{x}) = 0.0$



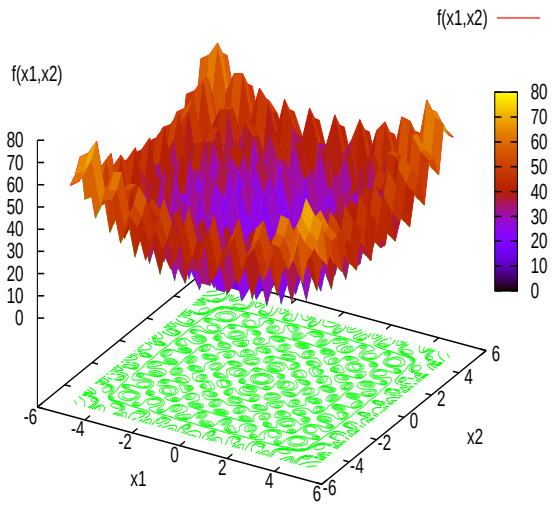


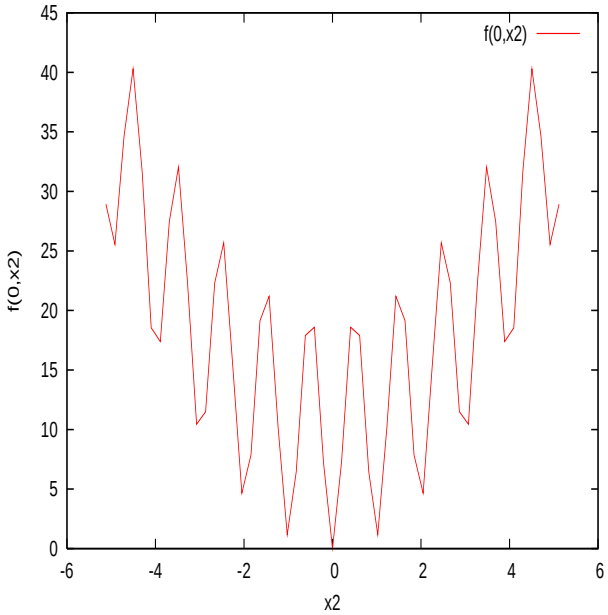
The Rastrigin Problem

Definition:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} (x_j^2 - 10\cos(2\pi x_j) + 10) \tag{10}$$

with $x_j \in [-5.12, 5.12]$ and $f^*(\mathbf{x}) = 0.0$





Constrained Optimzation Problem

Constrained minimization optimization problem $\mathbb{R}^n$*

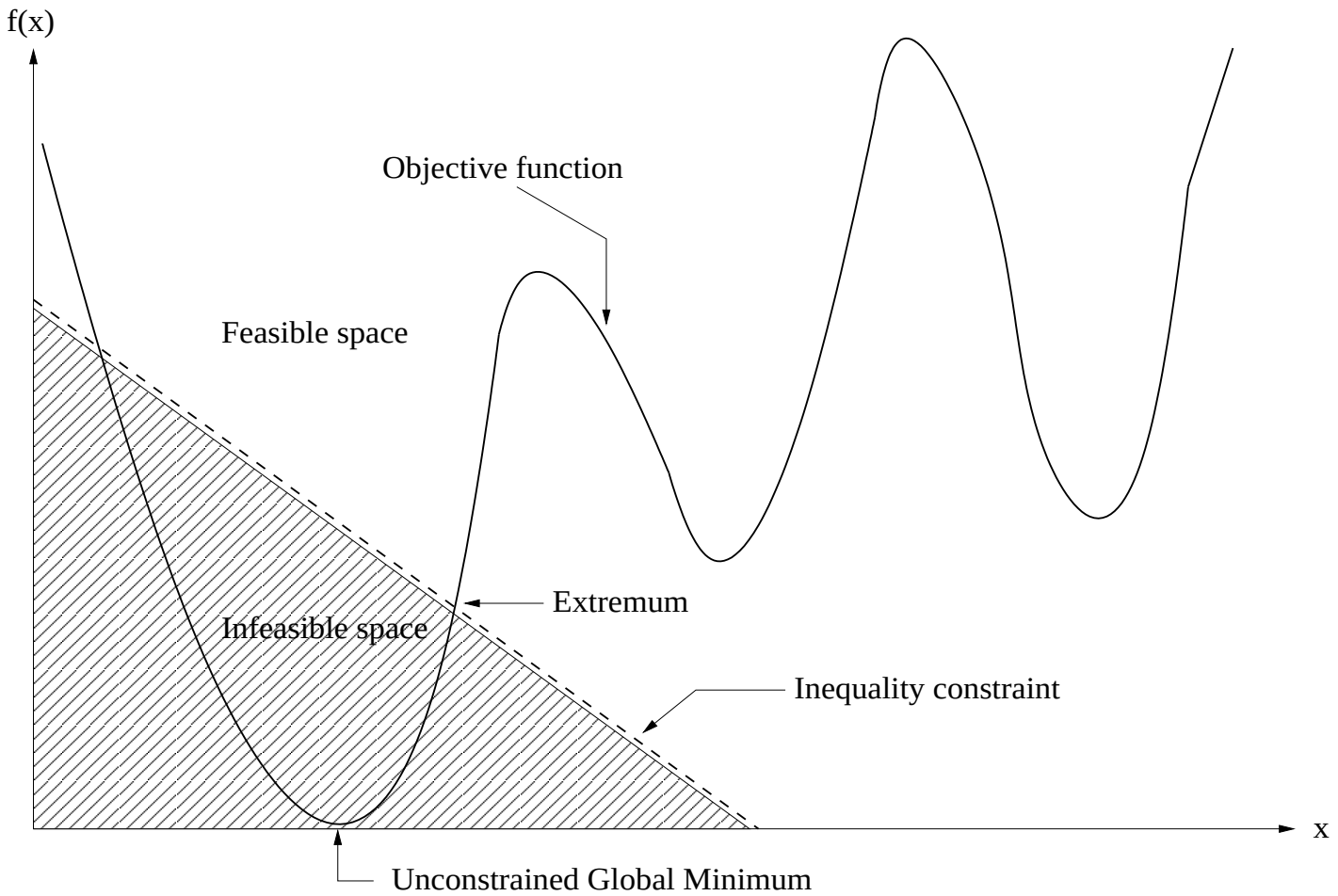
Given an objective function $f : \boldsymbol{D} \rightarrow \mathbb{R}$ and a set of constraints C, find a $\boldsymbol{x}^* \in \boldsymbol{D} \subset \mathbb{R}^n$ such that

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \boldsymbol{D} \tag{11}$$

and each $c \in C$, is satisfied by $\boldsymbol{x}^*$.

*C generally consists of inequality and equality constraints.

Constrained Optimzation Problem



Constrained Optimization Problem: Issues

The following new issues need to be considered when dealing with constrained optimization:

- How should two feasible solutions be compared?
- How should two infeasible solutions be compared?
  - should the infeasible solution with the best objective function value be preferred?
  - should the solution with the least number of constraint violations or lowest degree of violation be preferred?
  - should a balance be found between best objective function value and degree of violation?
- Should it be assumed that any feasible solution is better than any unfeasible solution? Alternatively, can objective function value and degree of violation be optimally balanced?

Constrained Optimization Problem

How do we deal with constrained optimization? One option is to convert to the problem into a simple one of direct optimization. Replace the original objective function f with

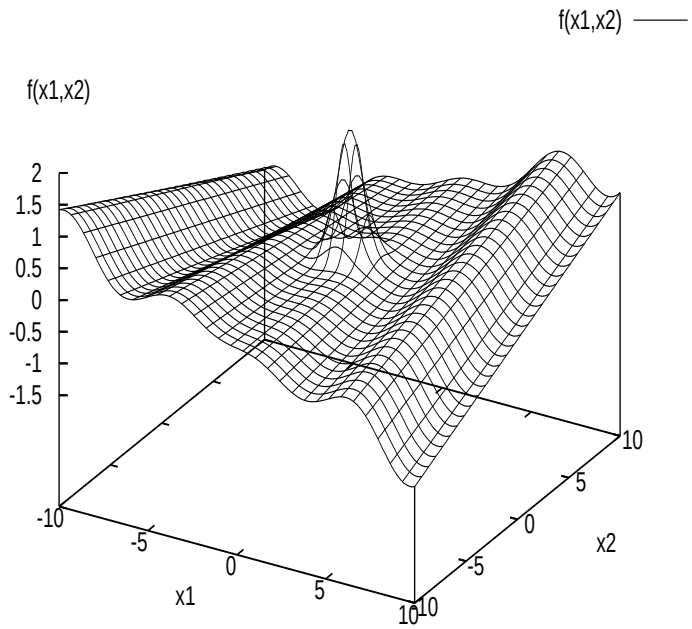
$$f^*(\mathbf{x}, t) = f(\mathbf{x}) + \lambda p(\mathbf{x}, t). \tag{12}$$

where λ is the penalty coefficient and $p(\mathbf{x}, t)$ is the (possibly) time-dependent penalty function.

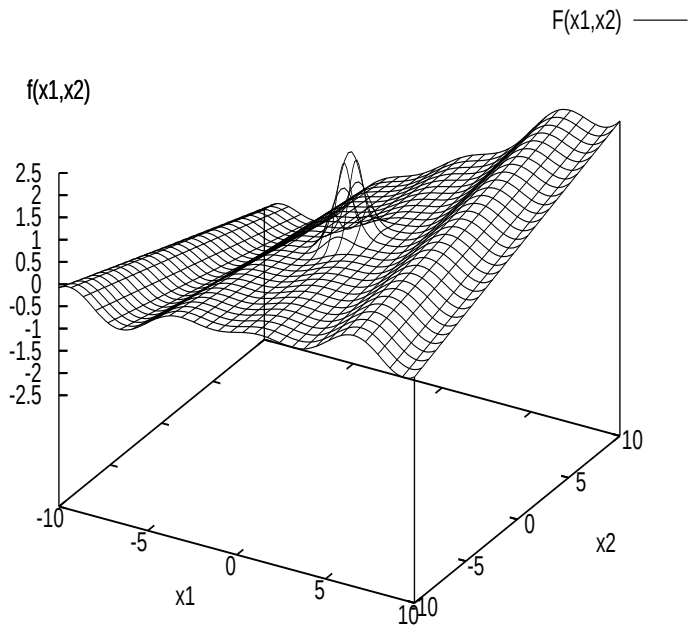
What are some of the problems with using this approach?

Constrained Optimization Problem: Effect of Penalty Methods

$$f(x_1, x_2) = \frac{x_1 \cos(x_1)}{20} + 2e^{-x_1^2-(x_2-1)^2} + 0.01x_1x_2$$



(e) Original function



(f) With penalty $p(x_1, x_2) = 3x_1$ and $\lambda = 0.05$

Constrained Optimization Problem: Convert to Unconstrained Problem

Convert the constrained (primal) problem to an unconstrained problem by defining the Lagrangian for the constrained problem:

$$L(\mathbf{x}, \lambda_g, \lambda_h) = f(\mathbf{x}) + \sum_{m=1}^{n_g} \lambda_{gm} g_m(\mathbf{x}) + \sum_{m=n_g+1}^{n_g+n_h} \lambda_{hm} h_m(\mathbf{x})$$

Where all constraints are rewritten to be ≤ 0 if satisfied. Then maximize the Lagrangian (dual problem):

$$\begin{array}{ll} \text{maximize}_{\lambda_g, \lambda_h} & L(\mathbf{x}, \lambda_g, \lambda_h) \\ \text{subject to} & \lambda_{gm} \geq 0, \quad m = 1, \ldots, n_g + n_h \end{array}$$

Constrained Optimization Problem: Convert to Unconstrained Problem (cont)

The vector $\mathbf{x}^*$ that solves the primal problem, as well as the Lagrange multiplier vectors, λ_g^* and λ_h^*, can be found by solving the min-max problem,

$$\min_{\mathbf{x}} \max_{\lambda_g, \lambda_h} L(\mathbf{x}, \lambda_g, \lambda_h)$$

Constrained Optimization Problem: Problem Examples

Constrained problem 1: Minimize the function

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

subject to the nonlinear constraints,

$$x_1 + x_2^2 \geq 0$$
$$x_1^2 + x_2 \geq 0$$

with $x_1 \in [-0.5, 0.5]$ and $x_2 \leq 1.0$.

The global optimum is $\mathbf{x}^* = (0.5, 0.25)$, with $f(\mathbf{x}^*) = 0.25$

Constrained Optimization Problem: Problem Examples (cont)

Constrained problem 2: Maximize the function

$$f(\mathbf{x}) = (\sqrt{n_x})^{n_x} \prod_{j=1}^{n_x} x_j$$

subject to the equality constraint,

$$\sum_{j=1}^{n_x} x_j^2 = 1$$

with $x_j \in [0, 1]$.

The solution is $\mathbf{x}^* = (\frac{1}{\sqrt{n_x}}, \ldots, \frac{1}{\sqrt{n_x}})$, with $f(\mathbf{x}^*) = 1$.

Dynamic Optimization Problems

Dynamic optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}, \varpi(t)), \quad \mathbf{x} = (x_1, \ldots, x_{n_x}), \varpi(t) = (\varpi_1(t), \ldots, \varpi_{n_\varpi}) \\ \text{subject to} \quad & g_m(\mathbf{x}) \leq 0, \ m = 1, \ldots, n_g \\ & h_m(\mathbf{x}) = 0, \ m = n_g + 1, \ldots, n_g + n_h \\ & x_j \in \text{dom}(x_j) \end{aligned}$$

where $\varpi(t)$ is a vector of time-dependent objective function control parameters. The objective is to find

$$\mathbf{x}^*(t) = \min_{\mathbf{x}} f(\mathbf{x}, \varpi(t))$$

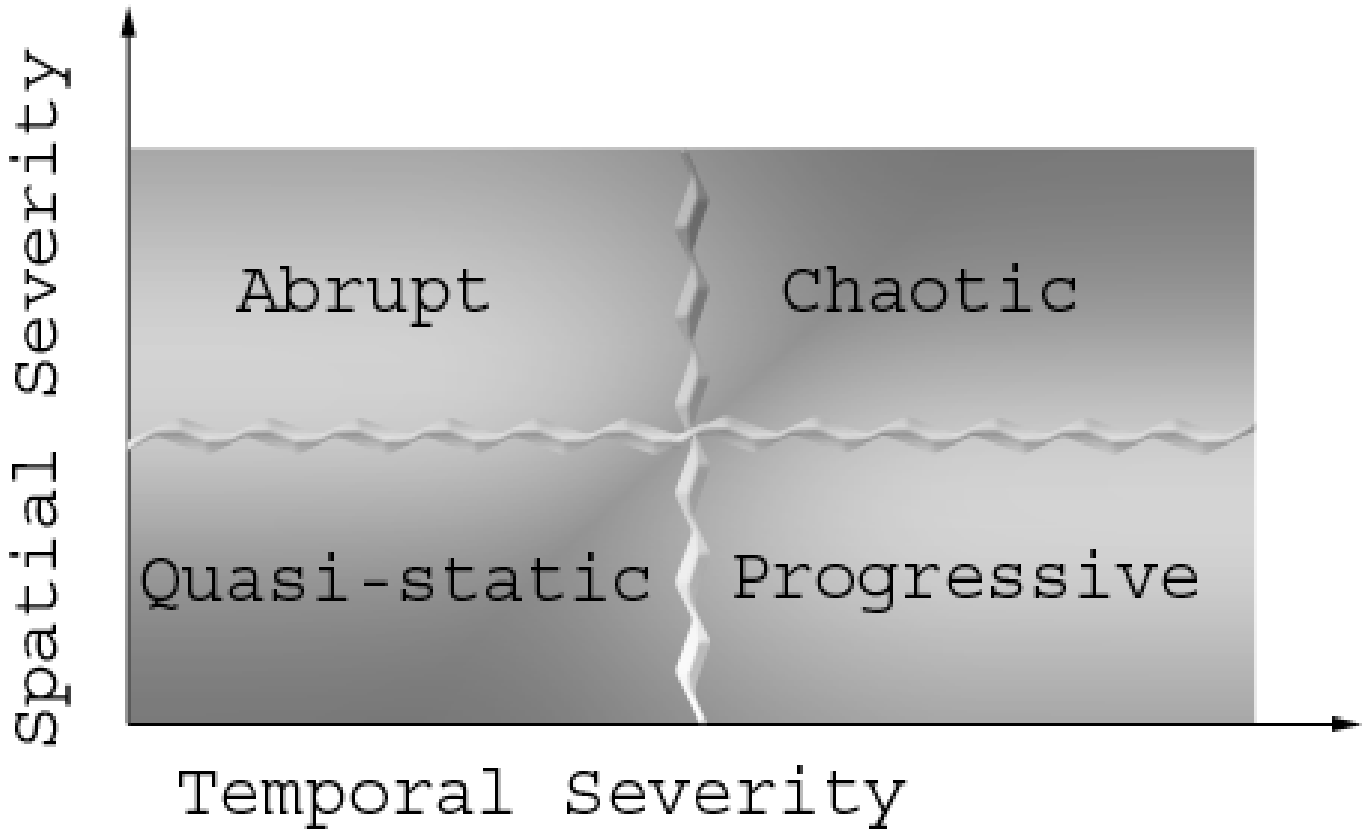
where $\mathbf{x}^*(t)$ is the optimum found at time step t

Dynamic Environment Types

- **Type I environments**, where the location of the optimum in problem space is subject to change. The change in the optimum, $\mathbf{x}^*(t)$ is quantified by the severity parameter, ζ, which measures the jump in location of the optimum.
- **Type II environments**, where the location of the optimum remains the same, but the value, $f(\mathbf{x}^*(t))$, of the optimum changes.
- **Type III environments**, where both the location of the optimum and its value changes.

Dynamic Environment Types

Change severity versus Change frequency

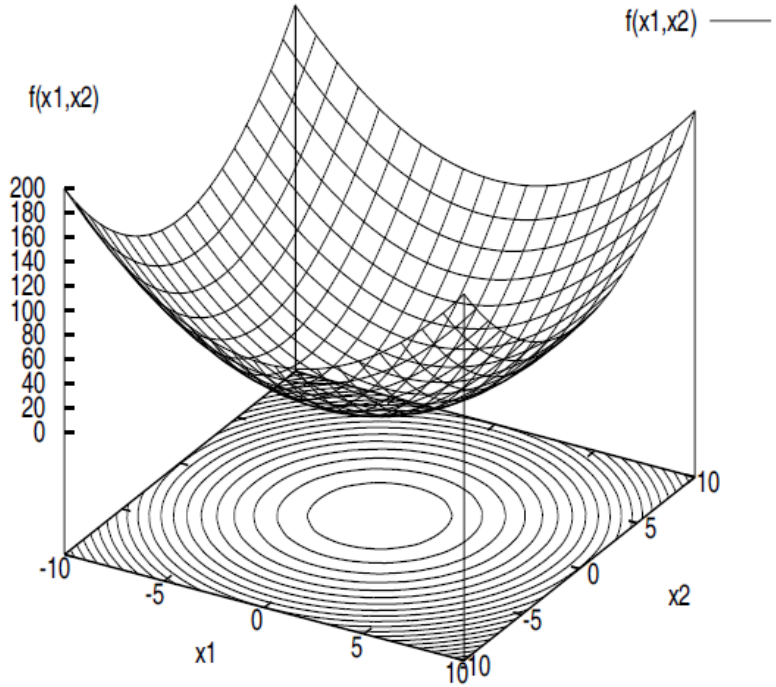


Dynamic Optimization Problem Example

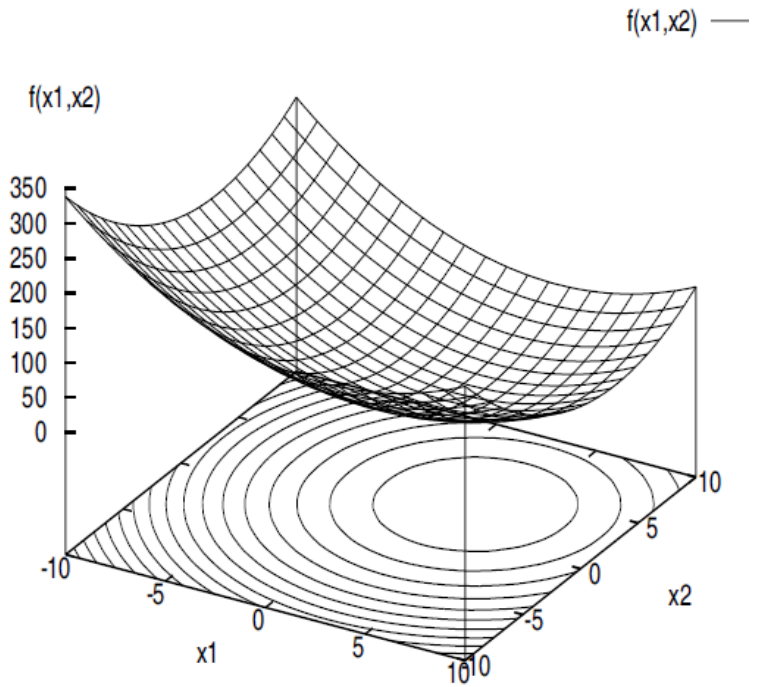
Consider:

$$f(\mathbf{x}, \varpi(\mathbf{t})) = \sum_{j=1}^{d} (x_j - \varpi_1(t))^2 + \varpi_2(t)$$

Dynamic Optimization Problem Example (cont)

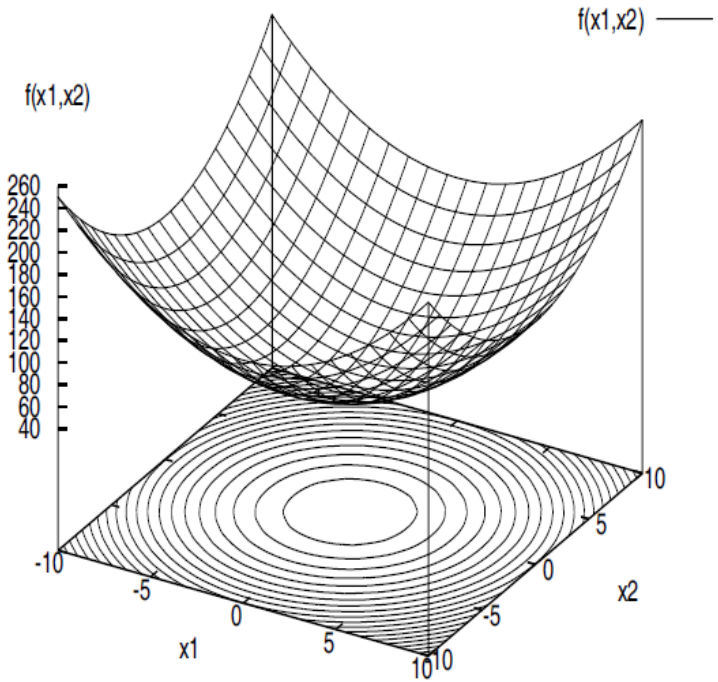


(a) Static function, $\varpi_1 = \varpi_2 = 0$

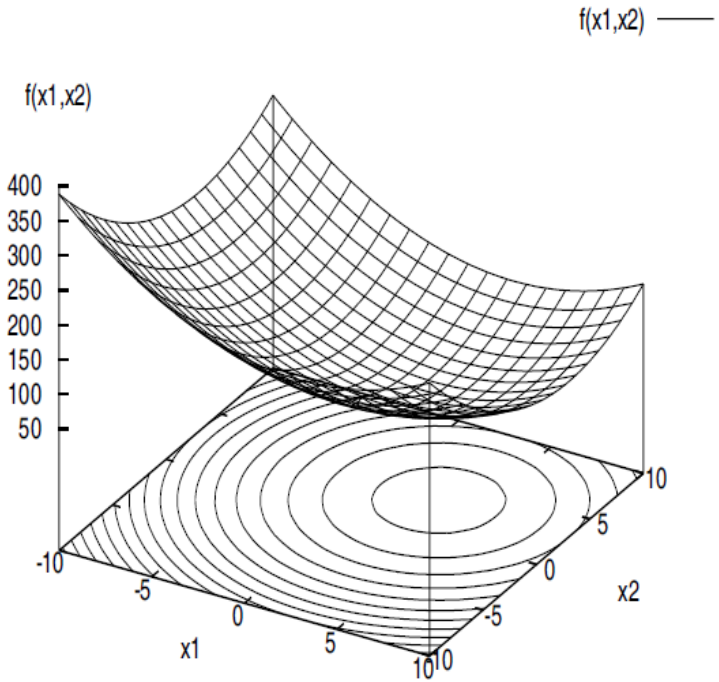


(b) Type I environment, $\varpi_1 = 3, \varpi_2 = 0$

Dynamic Optimization Problem Example (cont)



(c) Type II environment, $\varpi_1 = 0, \varpi_2 = 50$



(d) Type III environment, $\varpi_1 = 3, \varpi_2 = 50$

Multi-Objective Optimization

Multi-objective problem:

$$
\begin{array}{ll}
\text{minimize} & \mathbf{f}(\mathbf{x}) \\
\text{subject to} & g_m(\mathbf{x}) \leq 0, \quad m = 1, \ldots, n_g \\
& h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \ldots, n_g + n_h \\
& \mathbf{x} \in [\mathbf{x}_{min}, \mathbf{x}_{max}]^{n_x}
\end{array}
$$

where $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_{n_k}(\mathbf{x})) \in \mathcal{O} \subseteq \mathbb{R}^{n_k}$

$\mathcal{O}$ is referred to as the *objective space*

The search space, $\mathcal{S}$, is also referred to as the *decision space*

Multi-Objective Optimization: Meaning of an Optimum

- Imagine buying a car. You want to achieve the following goals:
  - minimise the cost
  - maximise the comfort
- These goals are in conflict with one another
  $\Rightarrow$ maximising the comfort increases the cost and vice versa
- Single solution does not exist $\Rightarrow$ Example of a MOOP

Multi-Objective Optimization: Meaning of an Optimum (cont)

The problem is the presence of conflicting objectives

Need to achieve a balance between these objectives

A balance is achieved when a solution cannot improve any objective without degrading one or more of the other objectives

There is not just one solution

Solutions are referred to as *non-dominated solutions*

Set of solutions is referred to as the Pareto-optimal set, and the corresponding objective vectors are referred to as the Pareto front

Multi-Objective Optimization: Weighted Aggregation Methods

Definition:

$$
\begin{array}{ll}
\text{minimize} & \sum_{k=1}^{n_k} \omega_k f_k(\mathbf{x}) \\
\text{subject to} & g_m(\mathbf{x}) \leq 0, \quad m = 1, \ldots, n_g \\
& h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \ldots, n_g + n_h \\
& \mathbf{x} \in [\mathbf{x}_{min}, \mathbf{x}_{max}]^{n_x} \\
& \omega_k \geq 0, k = 1, \ldots, n_k
\end{array}
$$

It is also usually assumed that $\sum_{k=1}^{n_k} \omega_k = 1$

Multi-Objective Optimization: Weighted Aggregation Methods (cont)

Aggregation methods have the following problems:

- The algorithm has to be applied repeatedly to find different solutions if a single-solution algorithm is used
- It is difficult to get the best weight values, ω_k, since these are problem-dependent

Multi-Objective Optimization: Pareto-Optimality

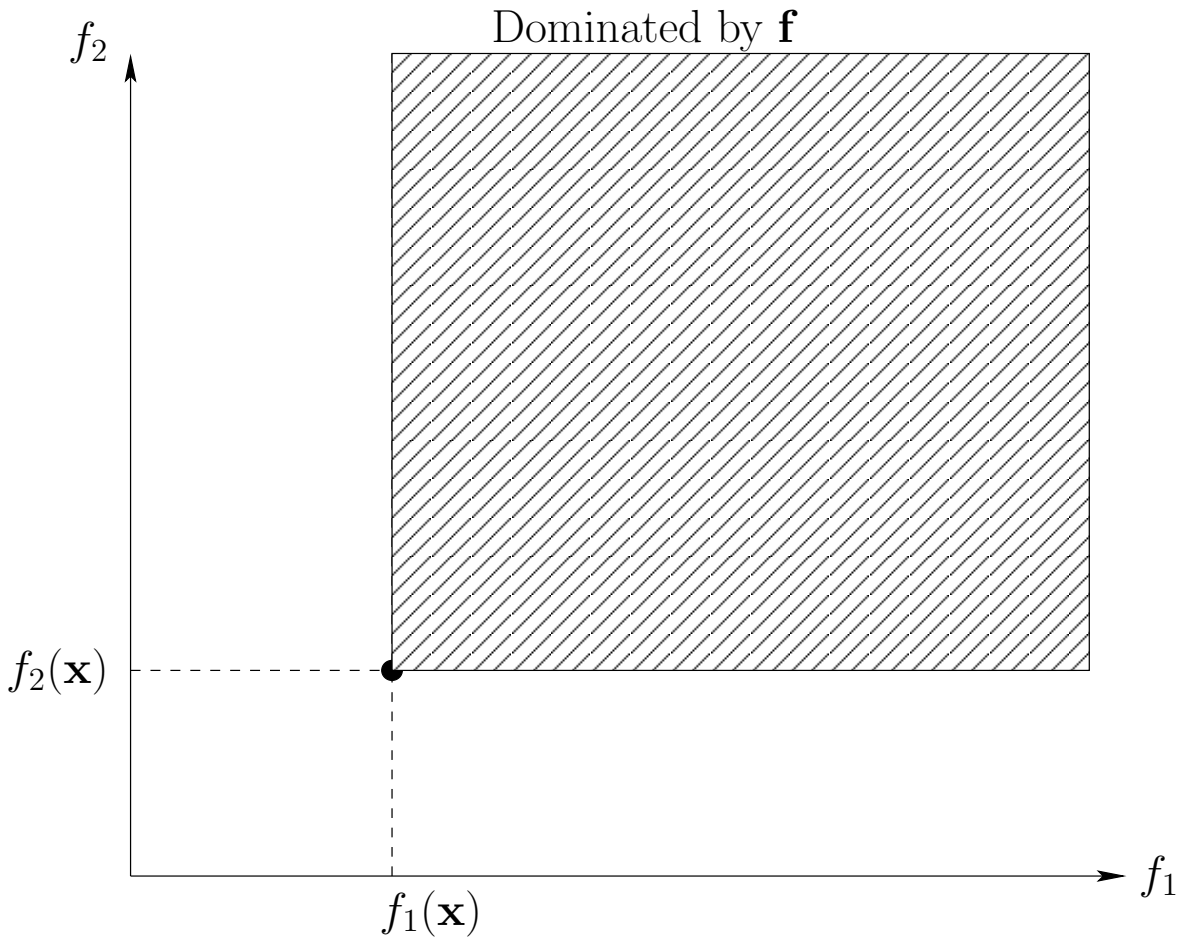
Domination:A decision vector, $\mathbf{x}_1$ dominates a decision vector, $\mathbf{x}_2$ (denoted by $\mathbf{x}_1 \prec \mathbf{x}_2$), if and only if

1. $\mathbf{x}_1$ is not worse than $\mathbf{x}_2$ in any objective, i.e. $f_k(\mathbf{x}_1) \leq f_k(\mathbf{x}_2), \forall k = 1, \ldots, n_k$, and
2. $\mathbf{x}_1$ is strictly better than $\mathbf{x}_2$ in at least one objective, i.e. $\exists k = 1, \ldots, n_k : f_k(\mathbf{x}_1) < f_k(\mathbf{x}_2)$.

So, solution $\mathbf{x}_1$ is better than solution $\mathbf{x}_2$ if $\mathbf{x}_1 \prec \mathbf{x}_2$ (i.e. $\mathbf{x}_1$ dominates $\mathbf{x}_2$), which happens when $\mathbf{f}_1 \prec \mathbf{f}_2$

- If only condition 1 is meet then a solution is said to be **weakly dominated**.

Multi-Objective Optimization: Pareto-Optimality (cont)



Multi-Objective Optimization: Pareto-Optimality (cont)

Pareto-optimal: *A decision vector, $\mathbf{x}^* \in \mathcal{F}$ is Pareto-optimal if there does not exist a decision vector, $\mathbf{x} \neq \mathbf{x}^* \in \mathcal{F}$ that dominates it. That is, $\nexists k : f_k(\mathbf{x}) < f_k(\mathbf{x}^*)$. An objective vector, $\mathbf{f}^*(\mathbf{x})$, is Pareto-optimal if $\mathbf{x}$ is Pareto-optimal.*

Pareto-optimal set: *The set of all Pareto-optimal decision vectors form the Pareto-optimal set, $\mathcal{P}^*$. That is,*

$$\mathcal{P}^* = \{\mathbf{x}^* \in \mathcal{F} | \ \nexists \mathbf{x} \in \mathcal{F} : \mathbf{x} \prec \mathbf{x}^*\}$$

Pareto-optimal front: *Given the objective vector, $\mathbf{f}(\mathbf{x})$, and the Pareto-optimal solution set, $\mathcal{P}^*$, then the Pareto-optimal front, $\mathcal{PF}^* \subseteq \mathcal{O}$, is defined as*

$$\mathcal{PF}^* = \{\mathbf{f} = (f_1(\mathbf{x}^*), f_2(\mathbf{x}^*), \ldots, f_k(\mathbf{x}^*)) | \mathbf{x}^* \in \mathcal{P}\}$$

Multi-Objective Optimization Problem Examples

With a convex, non-uniform Pareto front:

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= (f_1(\mathbf{x}), f_2(\mathbf{x})) \\ f_1(\mathbf{x}) &= x_1 \\ f_2(\mathbf{x}) &= g(\mathbf{x})(1 - \sqrt{f_1(\mathbf{x})/g(\mathbf{x})}) \end{aligned} \tag{13}$$

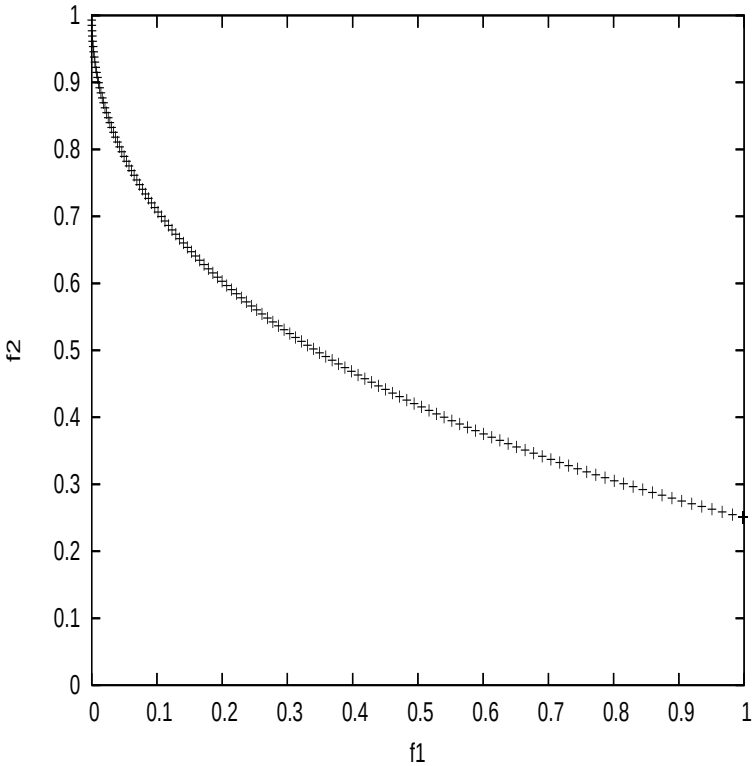
where

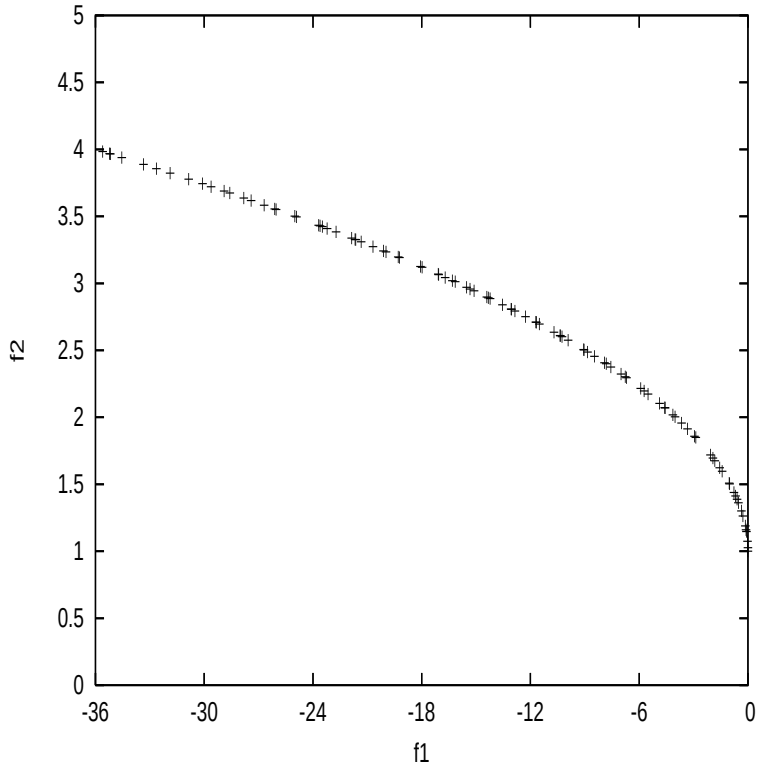
$$g(\mathbf{x}) = 1 + \frac{9}{n_x - 1} \sum_{j=2}^{n_x} x_j$$

With partially concave and partially convex Pareto front using the above equation, but with

$$f_2(\mathbf{x}) = g(\mathbf{x})(1 - \sqrt[4]{f_1(\mathbf{x})/g(\mathbf{x})} - (f_1(\mathbf{x})/g(\mathbf{x}))^4)$$

Multi-Objective Optimization Problem Examples(cont)





Niching, Multi-Solution Problems

What are multi-solution problems?

- Multi-solution problems generally are multi-modal, containing many optima.
- These optima may include more than one global optimum (i.e there is a tie) and a number of local minima

We generally attempt to solve multi-solution problems using niching/speciation algorithms

Niching, Multi-Solution Problems

Niching algorithms can be categorized based on the way that niches are located.

There are three primary categories:

- **Sequential niching** (or temporal niching) develops niches over time.
  - The process iteratively locates a niche, and removes any references to it from the search space.
  - Removal of references to niches usually involves modification of the search space.

Niching, Multi-Solution Problems

- **Parallel niching** locates all niches in parallel.
  - Individuals dynamically self-organize,or speciate, on the locations of optima.
- **Quasi-sequential niching** locates niches sequentially, but does not change the search space to remove the niche.

As a mental exercise consider finding the set of optima for a simple sinusoidal curve.