OMIS 6000

Week 5:

- **Optimization algorithms for nonlinear programs: newtons method, gradient-descent, and KKT conditions.**
- **Methods based on formulating the Lagrangian (penalty, barrier).**
- **Quadratic optimization problems.**

Adam Diamant, Schulich School of Business, Winter 2025

Solving Nonlinear Programs

In practice, nonlinear problems have thousands of constraints and decision variables. Analytical solutions (e.g., KKT) are typically not possible!

- **Nonconvex Problems:** One *cannot* guarantee that a solution is optimal. Use metaheuristics: particle swarm optimization or genetic algorithms.
- **Convex Problems:** Numerical methods (algorithms) are used to solve these problems which converge to the globally optimal solution.
  - Gradient-descent and projected gradient descent,
  - Penalty and interior-point methods.
  - And more (sequential quadratic programming)!

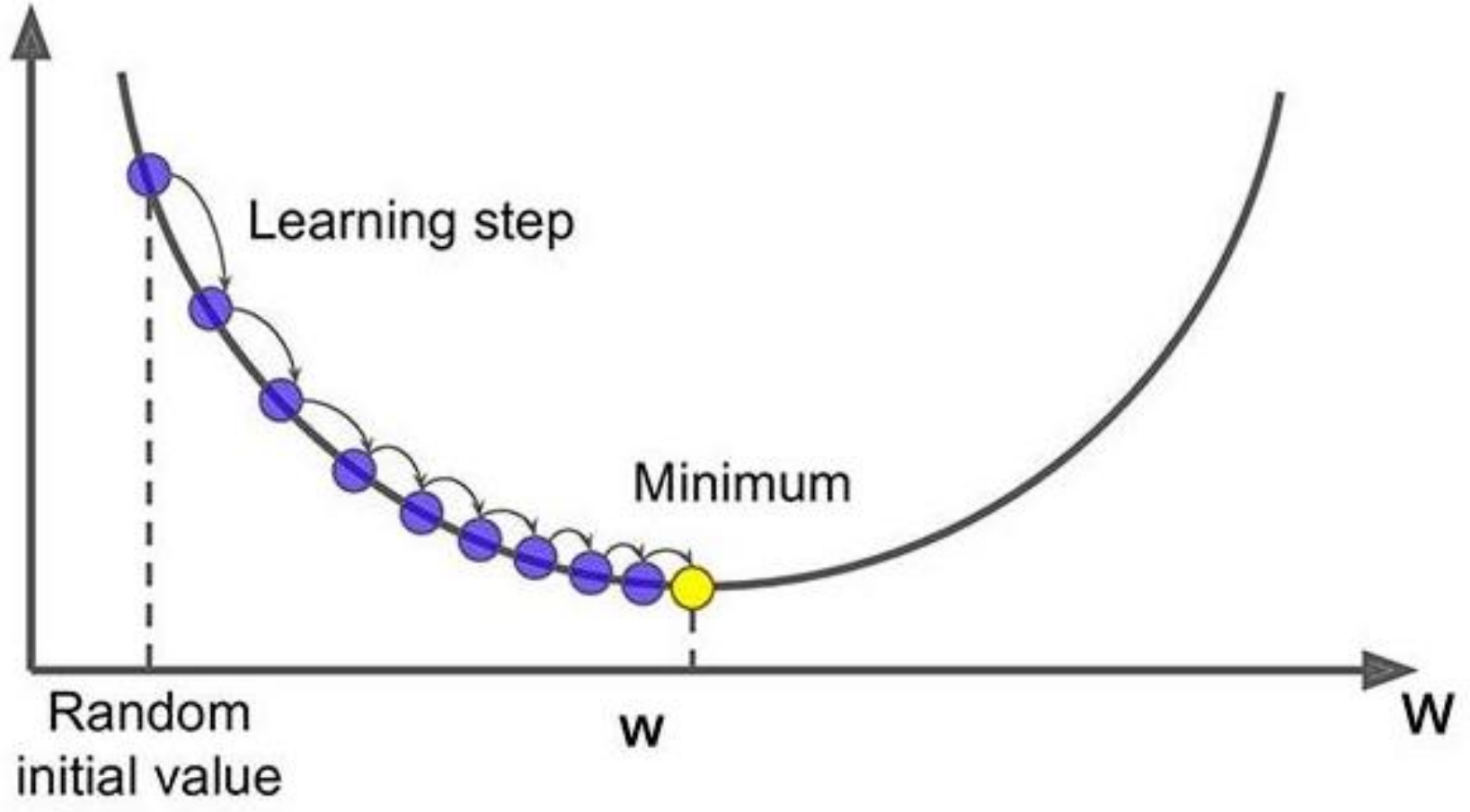
Solving Convex Problems

To solve nonlinear optimization problems, we are going to create algorithms that leverage at least one of two mathematical properties:

1. **Local Approximations:** By analyzing a small region around a point on a nonlinear function $\boldsymbol{f}$, we create a linear approximation that can be analyzed efficiently to iteratively obtain $\boldsymbol{x}^*$.
2. **Duality:** By analyzing the Lagrangian function and/or applying the KKT conditions, we can create algorithms that utilize this structure.

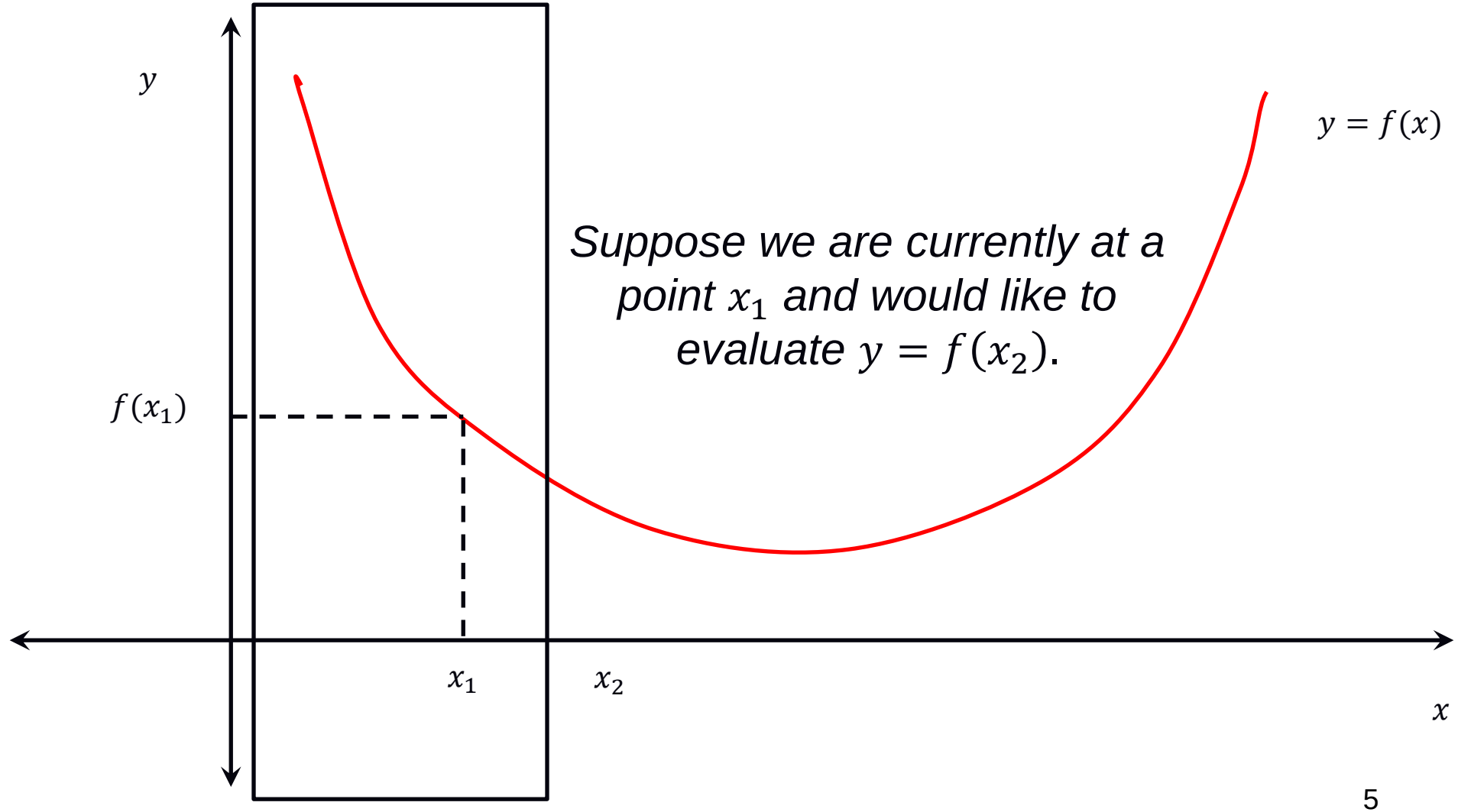
We first consider a convex function $f: \mathbb{R} \to \mathbb{R}$ and $x \in \mathbb{R}$.

Local Approximations



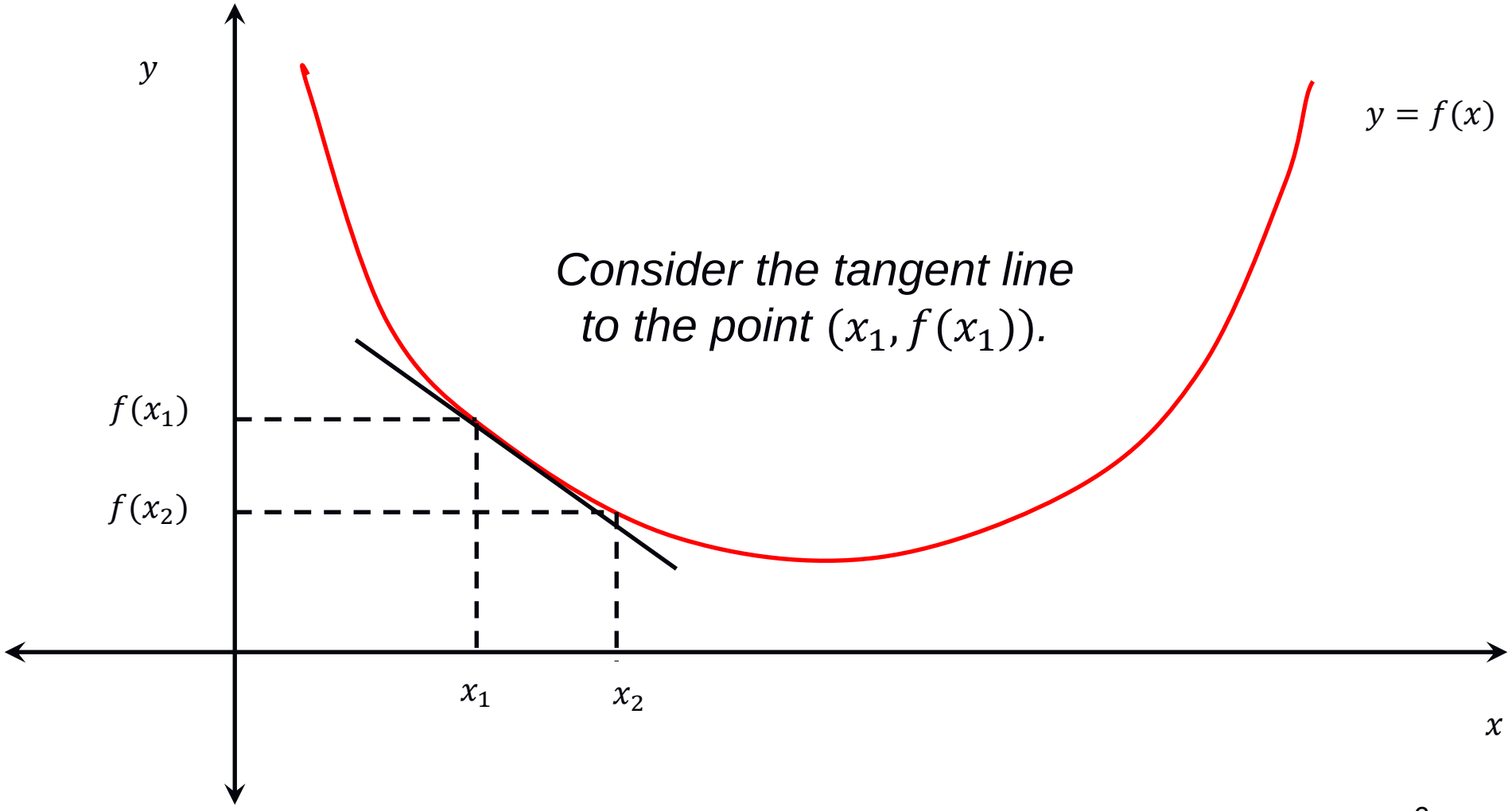
Source: https://towardsdatascience.com/quick-guide-to-gradient-descent-and-its-variants-97a7afb33add

Local Approximations

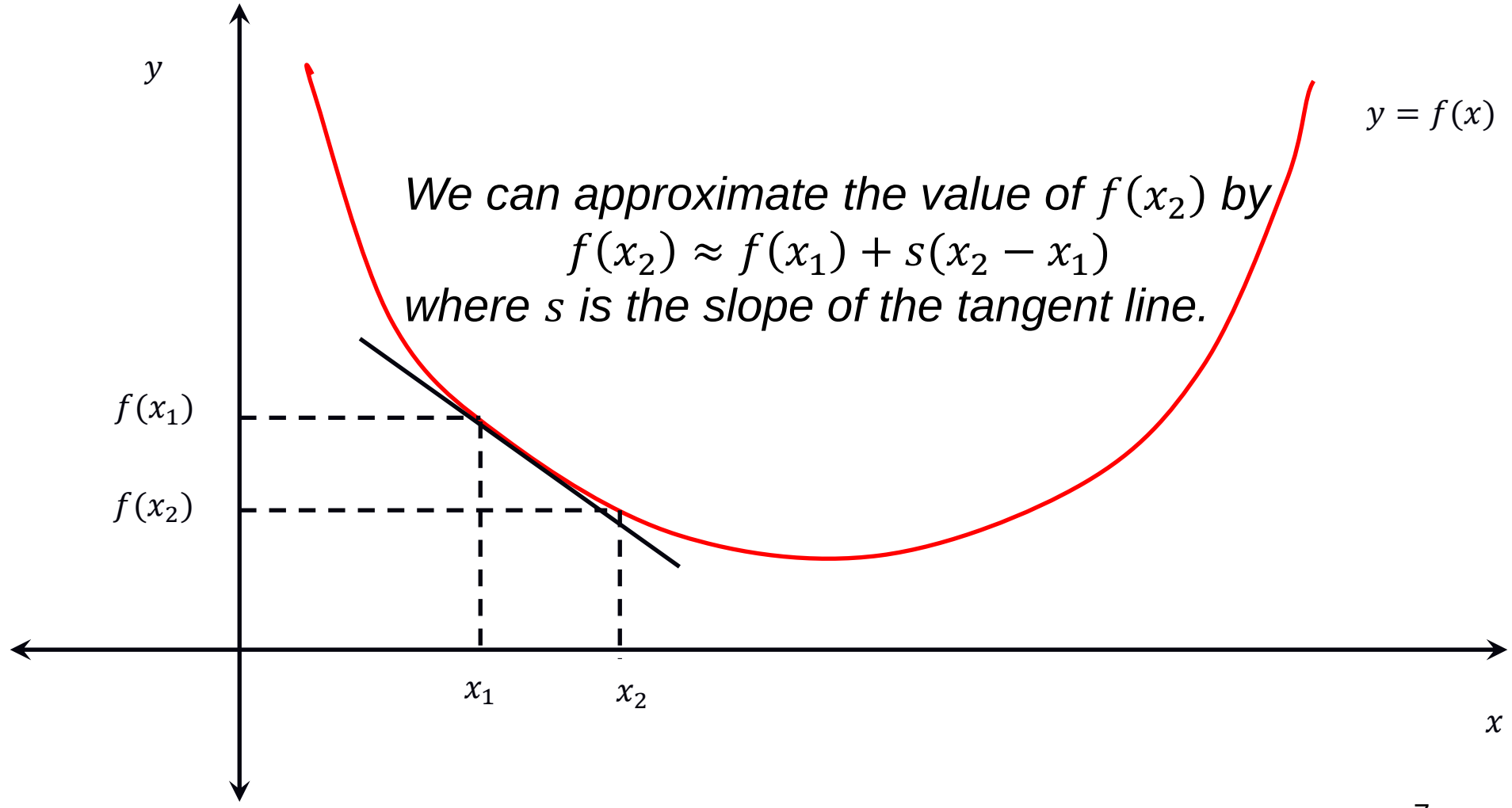


Suppose we are currently at a point x_1 and would like to evaluate $y = f(x_2)$.

Local Approximations



Local Approximations

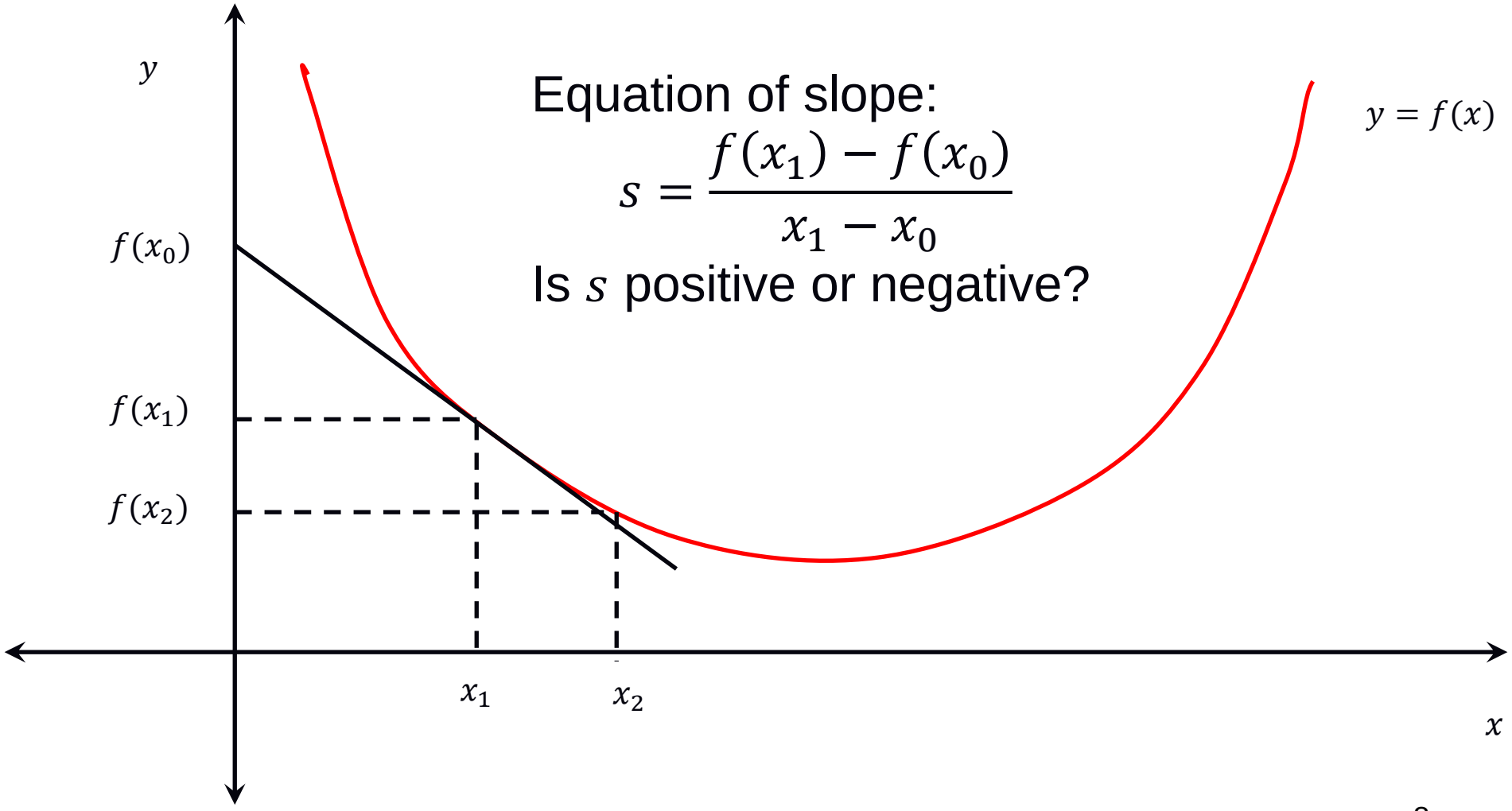


We can approximate the value of $f(x_2)$ by

$$f(x_2) \approx f(x_1) + s(x_2 - x_1)$$

where s is the slope of the tangent line.

Local Approximations

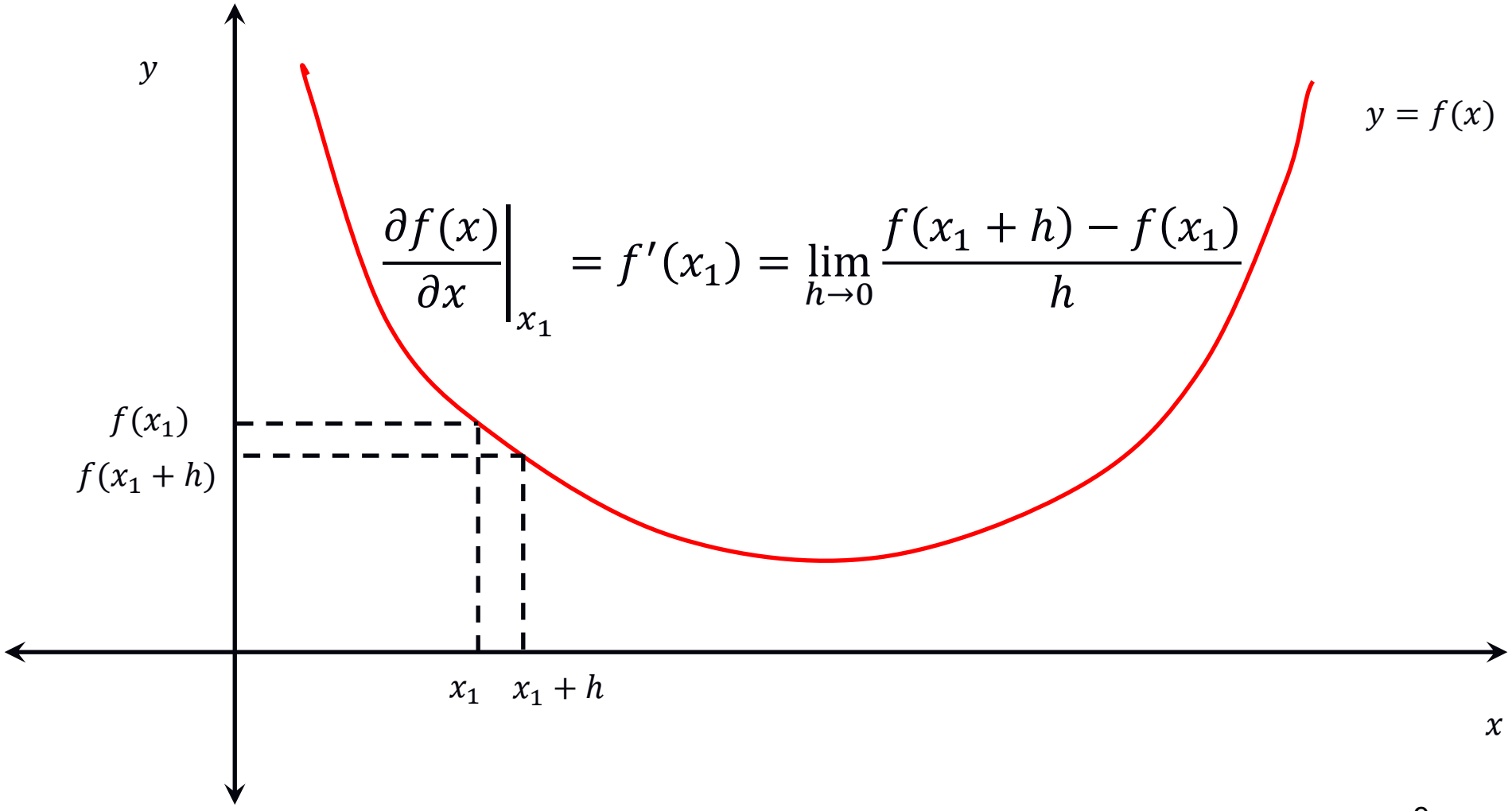


Equation of slope:

$$s = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Is s positive or negative?

Local Approximations



Local Approximations

First Derivative: The slope at a point.

$$\left.\frac{\partial f(x)}{\partial x}\right|_{x_1} = f'(x_1) = \lim_{h \to 0} \frac{f(x_1 + h) - f(x_1)}{h}$$

Second Derivative: The slope of the slope curve.

$$\left.\frac{\partial^2 f(x)}{\partial x^2}\right|_{x_1} = f''(x_1) = \lim_{h \to 0} \frac{f'(x_1 + h) - f'(x_1)}{h}$$

Local Approximations

- The first-order Taylor series expansion:

$$f(x_2) \approx f(x_1) + f'(x_1)(x_2 - x_1)$$

- It approximates the value of f at x_2. To make this more accurate, we can add another term.
- The second-order Taylor series expansion:

$$f(x_2) \approx f(x_1) + f'(x_1)(x_2 - x_1) + \frac{f''(x_1)}{2}(x_2 - x_1)^2$$

- If we keep adding terms, we get a better approximation to the true function. This comes at an increasingly higher computational cost…

Newton's Method

Newton's Method

1. Choose a starting solution x^0.
2. For iteration $k = 1, 2, 3, ...$
   a) If $\left|x^{k+1} - x^k\right| < \varepsilon$ then STOP.
   b) Solve $f'\left(x^k\right) + f''\left(x^k\right)\left(x^{k+1} - x^k\right) = 0$.

The idea is to perform a first-order Taylor expansion around the derivative of the function f. Because f is convex, a globally optimal solution exists for $f'\left(x^k\right) = 0$.

- Second derivatives may not exist, or it may be computationally expensive to calculate them for high dimensional problems.
- The initial solution could be poor leading to non-convergence.
- There are instances where the iterative algorithm leads to cycling.

Newton's Method

Example: Calculate the optimum of $f(x) = 3x^3 - x$.

Derivatives: $f'(x) = 9x^2 - 1$ and $f''(x) = 18x$.

Iterative Functions:

$$f'(x^k) + f''(x^k)(x^{k+1} - x^k) = 0$$
$$9(x^k)^2 - 1 + 18x^k(x^{k+1} - x^k) = 0$$
$$x^{k+1} = (9(x^k)^2 + 1)/18x^k$$

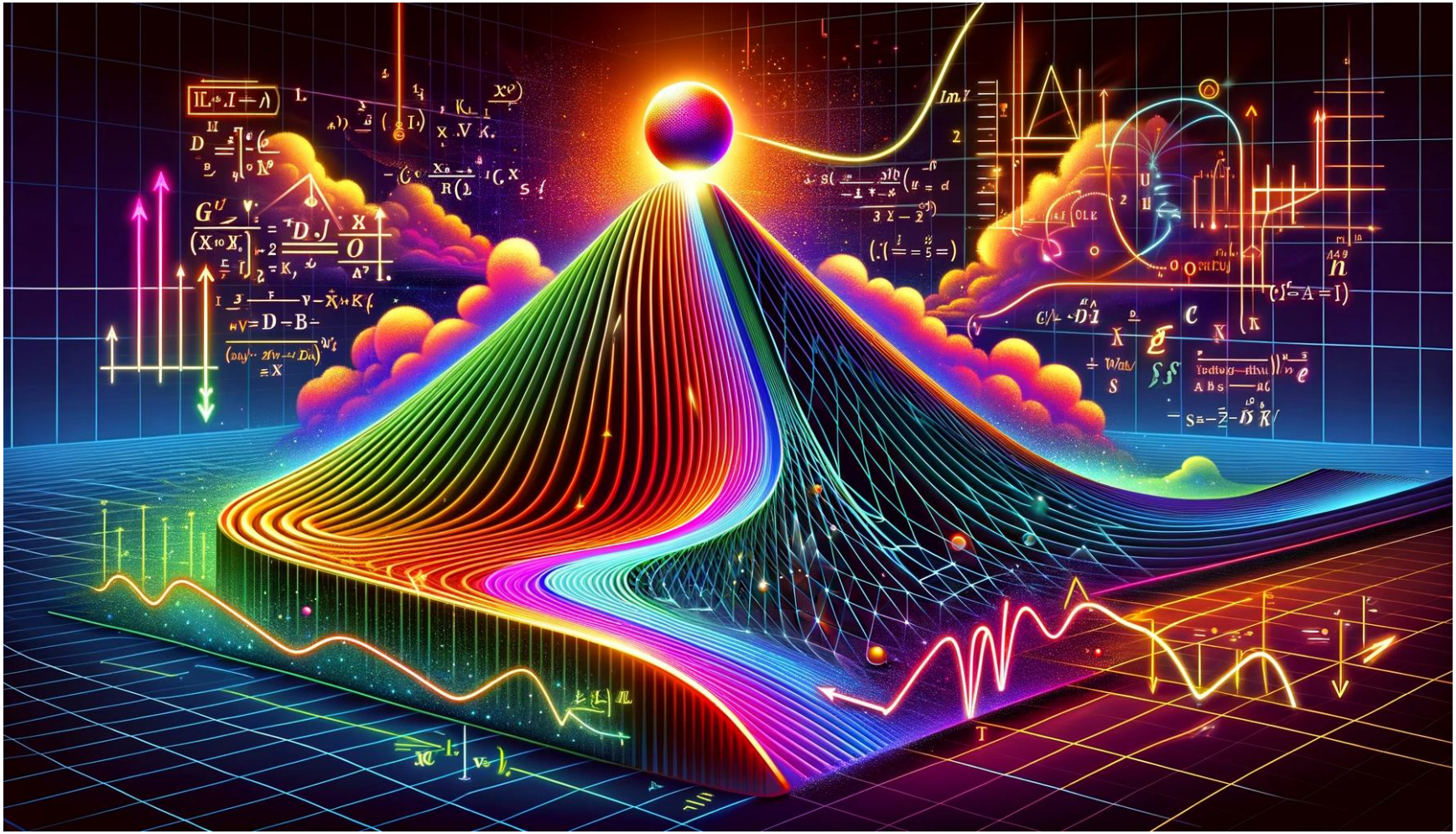
Sequence: $x^0 = 1$, $x^1 = 0.556$, $x^2 = 0.378$, $x^3 = 0.336$, $x^4 = 0.33334$, …, $x^{10} = 0.33333$.

Newton's Method

Theorem: To solve an optimization problem, a sequence of solutions $\{x^k\}$ are generated from an initial guess x^0 using second-order Taylor approximation of the objective function.

Proof: In-Class

Gradient Descent



Gradient Descent

1. Choose a starting solution x^0 and learning rate η (although this value can be adaptive).
2. For iteration $k = 1, 2, 3, ...$
   a) If $\left|x^{k+1} - x^k\right| < \varepsilon$ then STOP.
   b) Compute $x^{k+1} = x^k - \eta f'\left(x^k\right)$.

The idea is to find the minimum of a second-order Taylor expansion around f selecting $f''\left(x^k\right) = 1/\eta$ for $\eta \in \mathbb{R}$:

$$\min_y f\left(x^k\right) + f'\left(x^k\right)(y - x^k) + \frac{1}{2\eta}(y - x^k)^2$$

The next point $y = x^{k+1}$ is chosen to minimize the quadratic approximation by setting the gradient equal zero.

Gradient Descent

Example: Calculate the optimum of $f(x) = 3x^3 - x$.

Derivatives: $f'(x) = 9x^2 - 1$.

Iterative Functions:

$$x^{k+1} = x^k - \eta(9(x^k)^2 - 1)$$

Hyperparameter: Suppose I choose $\eta = 1/8$.

Sequence: $x^0 = 1$, $x^1 = 0$, $x^2 = 0.125$, $x^3 = 0.232$, $x^4 = 0.297$, $x^5 = 0.322$, …, $x^{10} = 0.3332$.

Gradient Descent

Theorem: To solve an optimization problem, a sequence of solutions $\{x^k\}$ are generated from an initial guess x^0 by minimizing a second-order Taylor approximation of the function f with a step size selected to be $f''\left(x^k\right) = 1/\eta$.

Proof: In-Class

Generalization Questions

We have seen examples of optimization algorithms for 2-D problems. How can we:

1. Generalize these algorithms to problems with multidimensional variables?

2. Apply the algorithms when optimizing an objective function over a constraint set $\mathcal{C}$?

Duality

Exploiting the Lagrangian

Consider the following nonlinear, but convex, optimization problem where $\boldsymbol{g}: \mathbb{R}^n \rightarrow \mathbb{R}^m$, $\boldsymbol{h}: \mathbb{R}^n \rightarrow \mathbb{R}^k$ and $f: \mathbb{R}^n \rightarrow \mathbb{R}$ for $\boldsymbol{x} \in \mathbb{R}^n$.

$$z^* = \min_{\boldsymbol{x} \geq \mathbf{0}} f(\boldsymbol{x}) \;\; s.t. \; \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \; and \; \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}.$$

We assume that f is a convex function and

$$\mathcal{C} = \{\boldsymbol{x} \geq \mathbf{0} : \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \; and \; \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}\}$$

is a convex constraint set.

Exploiting the Lagrangian

We can formulate an unconstrained optimization problem where $\boldsymbol{g}: \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{h}: \mathbb{R}^n \to \mathbb{R}^k$, $f: \mathbb{R}^n \to \mathbb{R}$ for $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{\lambda} \in \mathbb{R}^m_+$ and $\boldsymbol{\mu} \in \mathbb{R}^k$.

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{h}(\boldsymbol{x})$$

where $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ is the **Lagrangian** and the variables $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ **Dual/Lagrange Multipliers**.

Exploiting the Lagrangian

Since the **Lagrangian** is *unconstrained*, it is easier to search for the optimal solution because the feasible region does not include complicating constraints.

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{h}(\boldsymbol{x})$$

There are drawbacks:

- Convergence issues due to the large search area.
- No ability to decompose the problem or to use special structure to devise performant algorithms.

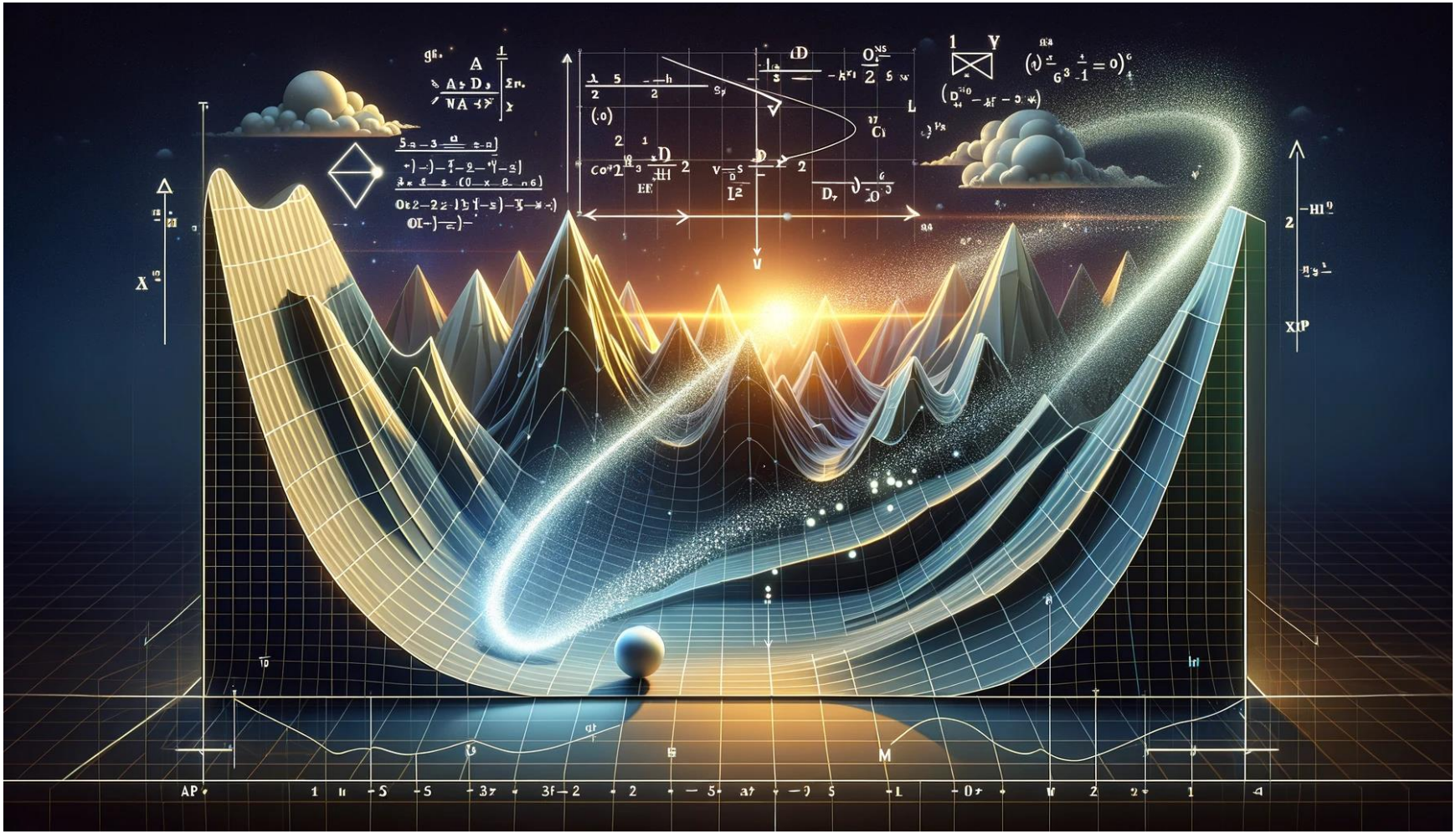
Algorithms for Constrained Optimization

Projected Gradient Descent

Lagrangian Penalty Method

Barrier Functions

Projected Gradient Descent



Projected Gradient Descent

Idea: While vanilla gradient descent is not necessarily the best optimization technique in terms of convergence speed, it is pervasively used in practice for high-dimensional problems.

1. Each update step is simple to compute.
2. Global convergence (for convex problems) can be guaranteed with relatively few assumptions.
3. Implementing the algorithm is easy. It rarely requires problem specific logic.

Can we adapt the univariate algorithm for multivariate constrained optimization problems?

Multidimensional Problems

First Derivative (Gradient): The slope at a point.

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \dots, \frac{\partial f(\boldsymbol{x})}{\partial x_m} \right)$$

Second Derivative (Hessian): The slope of the slope curve.

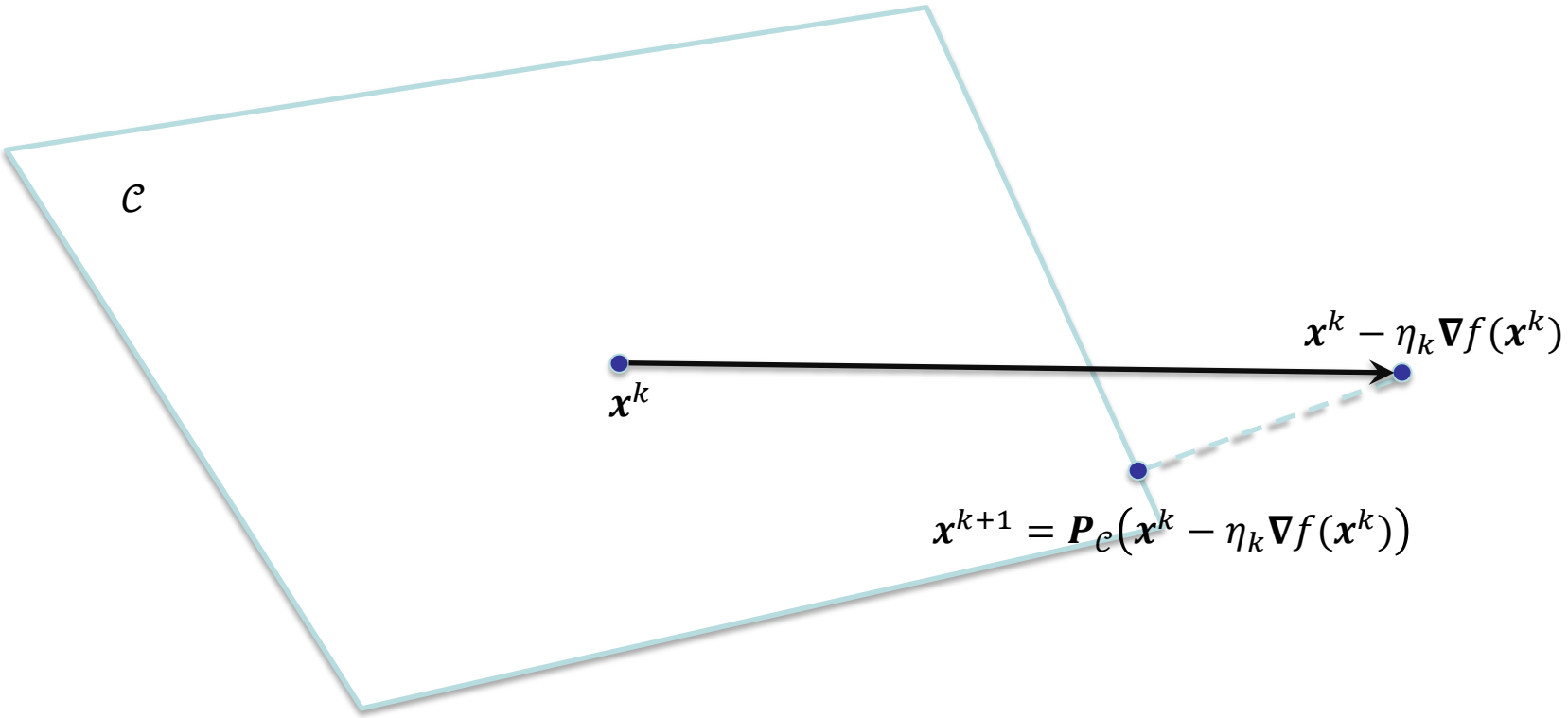
$$\nabla^2_{xx} f(\boldsymbol{x}) = \boldsymbol{H}(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_m} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_m} \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_m \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_m \partial x_2} \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_m^2} \end{pmatrix}$$

Projected Gradient Descent

1. Choose a starting solution $\boldsymbol{x}^0$.
2. For iteration $k = 1, 2, 3, ...$
   a) If $\left\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\right\| < \varepsilon$ then **STOP**.
   b) Choose an appropriate step size η_k.
   d) Update $\boldsymbol{x}^{k+1} = \boldsymbol{P}_{\mathcal{C}}\left(\boldsymbol{x}^k - \eta_k \nabla f(\boldsymbol{x}^k)\right)$.

There are many strategies (e.g., adaptive) for choosing the step size η_k. Often, one chooses $\eta_k = \eta \in \mathbb{R}$ for all k noting that if η is too small, convergence takes a long time while choosing η to be too large a value leads to divergence.

Projected Gradient Descent



Examples of Projections

- Project on to $\mathcal{C} = \{\boldsymbol{x}: \boldsymbol{x} \leq 0\}$?
  - $\text{proj}_{\mathcal{C}}(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}_i, \text{if } \boldsymbol{x}_i \leq 0 \\ 0, \text{otherwise} \end{cases}$
- Project on to $\mathcal{C} = \{\boldsymbol{x}: \|\boldsymbol{x}\|_2 \leq 1\}$?
  - $\text{proj}_{\mathcal{C}}(\boldsymbol{x}) = \begin{cases} \boldsymbol{x}, \text{if } \|\boldsymbol{x}\|_2 \leq 1 \\ \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}, \text{otherwise} \end{cases}$

Examples of Projections

The projection step $\boldsymbol{x}^{k+1} = \boldsymbol{P}_{\mathcal{C}}\left(\boldsymbol{x}^k - \eta_k \nabla f(\boldsymbol{x}^k)\right)$.

can be formulated as an optimization problem!

Define $\boldsymbol{y} = \boldsymbol{x}^k - \eta_k \nabla f(\boldsymbol{x}^k)$ to be the *unprojected* application of gradient descent. Then, solve

$$\min_{\boldsymbol{x}^{k+1} \in \mathcal{C}} \left\| \boldsymbol{x}^{k+1} - \boldsymbol{y} \right\|^2$$

If the constraint set is linear, the problem is a quadratic program which is a relatively simple nonlinear problem to solve.

- You can even use the KKT conditions!

Projected Gradient Descent

- **Advantages**:
  - Algorithmically, there is only one additional step on top of vanilla gradient descent.
  - It is fast for simple constraints, e.g., $x \geq 0$.
  - Theoretically, it gives the same convergence guarantees as univariate gradient descent (if the projection can be computed efficiently).
- **Disadvantages**:
  - Slow/inaccurate for complicated constraints..

Price Optimization Example Revisited: PGD Approach

Maximize $\mathbf{Z} = p_1(35234 - 26p_1) + p_2(27790 - 9p_2)$

Subject to:

$35234 - 26p_1 \geq 0$ (Demand constraint #1)

$27790 - 9p_2 \geq 0$ (Demand constraint #2)

$p_2 \geq 550 + p_1$ (Pricing constraint)

$p_1, p_2 \geq 0$ (Nonnegativity constraints)

If we solve using Gurobi, the optimal solution is:

$$p_1 = \$677.57, p_2 = \$1543.88$$

Let's confirm using the projected gradient descent.

Price Optimization Example Revisited: PGD Approach

Step 1: Setup the parameters

- Choose a stopping criterion $\varepsilon = 1e^{-6}$.
- Choose a step size $\eta_1 = \eta_2 = -0.001$.

Step 2: Vanilla Gradient Descent $(\boldsymbol{x}^k - \eta_k \nabla f(\boldsymbol{x}^k))$

$$\tilde{p}_1^{k+1} = p_1^k + \eta_1\left(35234 - 2 \times 26 p_1^k\right)$$

$$\tilde{p}_2^{k+1} = p_2^k + \eta_2\left(27790 - 2 \times 9 p_1^k\right)$$

Price Optimization Example Revisited: PGD Approach

Step 3: Solve a quadratic program to project the unprojected gradient descent step onto the constraints.

Minimize $\mathbf{Z} = \left(p_1^{k+1} - \tilde{p}_1^{k+1}\right)^2 + \left(p_2^{k+1} - \tilde{p}_2^{k+1}\right)^2$

Subject to:

$35234 - 26p_1^{k+1} \geq 0$ (Demand constraint #1)

$27790 - 9p_2^{k+1} \geq 0$ (Demand constraint #2)

$p_2^{k+1} \geq 550 + p_1^{k+1}$ (Pricing constraint)

$p_1^{k+1}, p_2^{k+1} \geq 0$ (Nonnegativity constraints)

Price Optimization Example Revisited: PGD Approach

Keep iterating until the **stopping criteria** holds:

$$\left(p_1^{k+1} - p_1^k\right)^2 + \left(p_2^{k+1} - p_2^{k+1}\right)^2 \leq \varepsilon$$

- Typically, it will take many iterations to converge. Thus, it is important that each step of the algorithm is computationally efficient.
- Why is this not the best example? What changes would you make to the problem in order to better justify the use of PGD?

Penalty Methods

Lagrangian Penalty Method

Idea: Use the **Lagrangian** function.

- By **dualizing** the constraints, the nonlinear optimization problem becomes *unconstrained*.
- Unconstrained optimization problems are amenable to optimization methods like Newton's Method or Gradient Descent.
- To do this, we modify the Lagrangian:

$$\hat{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{\varphi}(\boldsymbol{g}(\boldsymbol{x})) + \boldsymbol{\mu}^T \boldsymbol{\phi}(\boldsymbol{h}(\boldsymbol{x}))$$

by incorporating **penalty** functions $\boldsymbol{\varphi}(\cdot)$ and $\boldsymbol{\phi}(\cdot)$.

Lagrangian Penalty Method

Penalty functions are chosen by the user.

- Should be convex and monotonically increasing.
- Slope is increasingly steep for infeasible points

$$\lim_{t \to \infty} \varphi'(t) = \infty .$$

- Slope is increasingly flat for feasible points

$$\lim_{t \to -\infty} \varphi'(t) = 0 .$$

- At zero, we should have $\varphi(0) = 0$.

Examples: $e^x - 1,\ \max\{0, g(x)\}^2,\ h(x)^2$

Lagrangian Penalty Method

1. Select a set of penalty functions $\boldsymbol{\varphi}$ and $\boldsymbol{\phi}$.
2. Choose a starting dual solution $(\boldsymbol{\lambda}^0, \boldsymbol{\mu}^0)$.
3. For $k = 1, 2, 3, \ldots$
   a) Solve the following optimization problem:

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) + \boldsymbol{\lambda}^k \boldsymbol{\varphi}(\boldsymbol{g}(\boldsymbol{x})) + \boldsymbol{\mu}^k \boldsymbol{\phi}(\boldsymbol{h}(\boldsymbol{x}))$$

   using Newton's Method or Gradient Descent.

   a) If the optimal $\boldsymbol{x}^*$ is feasible then **STOP**.

   b) Otherwise, choose:

      a) $\boldsymbol{\lambda}^{k+1} \geq \boldsymbol{\lambda}^k$ where one dimension is strict.

      b) $\boldsymbol{\mu}^{k+1} \geq \boldsymbol{\mu}^k$ where one dimension is strict.

Barrier Functions

Idea: Penalty functions penalize *infeasibility*. Instead, can we use functions that remain finite only when solutions $\boldsymbol{x}$ are feasible?

- By **dualizing** the constraints, the nonlinear optimization problem becomes *unconstrained*.
- Unconstrained optimization problems are amenable to optimization methods like Newton's Method or Gradient Descent (for decision vectors $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$).
- To do this, we modify the Lagrangian

$$\hat{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \lambda\varphi\big(\boldsymbol{g}(\boldsymbol{x})\big) + \mu\phi(\boldsymbol{h}(\boldsymbol{x}))$$

by incorporating **barrier** functions $\varphi(\cdot)$ and $\phi(\cdot)$.

Barrier Functions

Barrier functions are chosen by the user.

- Should be convex and monotonically increasing.
- Returns infinite values for infeasible points

$$\varphi\,(\boldsymbol{x}) = \infty \;\; for \;\; \boldsymbol{x} \notin \mathcal{C}.$$

- Returns feasible values for feasible points

$$\varphi\,(\boldsymbol{x}) < \infty \;\; for \;\; \boldsymbol{x} \in \mathcal{C}\,.$$

- The function should return larger values the closer we are to the boundary of the feasible region, and smaller values the closer we are to the middle or "center" of the feasible region.

Examples: $log(x)\,,\;\; 1/x\,,\;\; log(ax + bx + c)$

Barrier Functions

1. Select a set of barrier functions φ and ϕ.
2. Choose a starting dual solution (λ^0, μ^0).
3. For $k = 1, 2, 3, \ldots$
   a) Solve the following optimization problem:

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) + \lambda^k \varphi\big(\boldsymbol{g}(\boldsymbol{x})\big) + \mu^k \phi(\boldsymbol{h}(\boldsymbol{x}))$$

   using Newton's Method or Gradient Descent.

   a) If the optimal $\boldsymbol{x}^*$ is feasible then **STOP**.

   b) Otherwise, choose:

      a) $\lambda^{k+1} \in (0, \lambda^k)$

      b) $\mu^{k+1} \in (0, \mu^k)$

Penalty and Barrier Methods

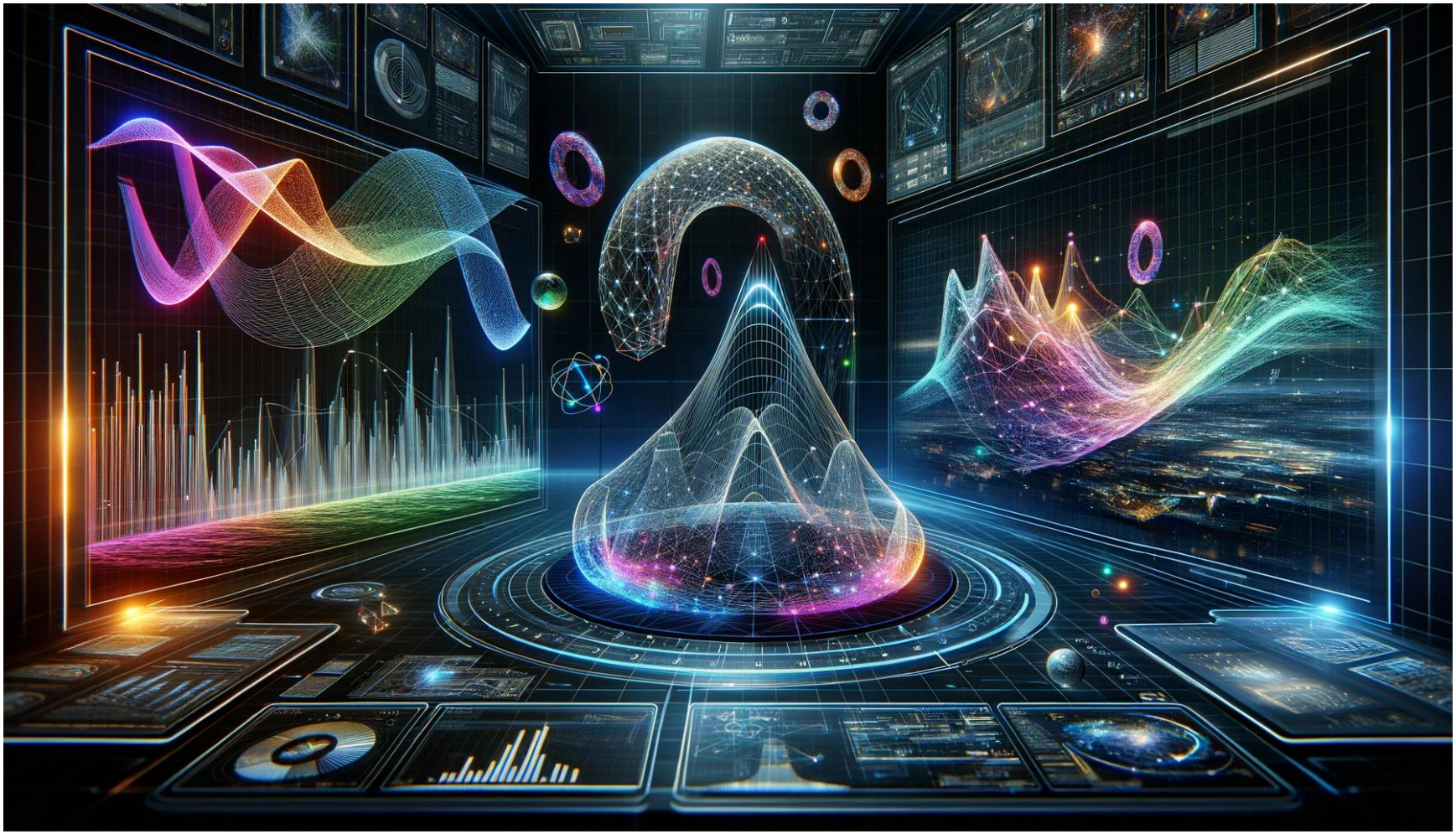
We have complete freedom:

1. Create one penalty function per constraint and then one dual multiplier for each constraint.
2. Create a single penalty function by aggregating all the constraints into a single function and then use a single dual multiplier for this function.

The decision depends on the problem type and your experience (it's more art than a science).

- **The difference**: The Lagrangian Penalty Method progressively *increases* the penalty for **infeasibility**. The Barrier Method progressively *decreases* the penalty for **infeasibility**. They are very similar…

What method does Gurobi use?



Barrier Methods

Nonlinear Programming Problems (NLPs)

There are many other approaches; our discussion represents the tip of the iceberg.

- While these techniques are intricate, they typically cannot solve problems of the same magnitude as linear programs (LPs).
- For many nonlinear programming models, it is difficult to even find a good approximation.
- Non-convex problems are even more difficult to solve and require heuristic methods such as particle swarm optimization or genetic algorithms. No guarantee of optimality!

Nonlinear Programming Problems (NLPs)

Given this numerical complexity, why do we need to formulate nonlinear programs?

- There are many applications in engineering, supply chain management, and political/economic planning that, to *faithfully* model the real-world system, require a nonlinear objective and/or constraint set.
- They are the bedrock for designing algorithms used to train machine learning models.
- Algorithmic knowledge can help improve performance (e.g., subgradient descent).

Nonlinear Programming Problems (NLPs)

Ridge Regression

Variable Pricing with Diversion

Markdown Optimization

Ridge Regression

Source: *https://en.wikipedia.org/wiki/Ridge*

Ridge Regression

Dataset: You have a data set of $i = 1, \ldots, N$ instances with $j = 1, \ldots, J$ features x_{ij} for every outcome y_i.

$$\textbf{Minimize } Z = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \alpha - \sum_{j=1}^{J} \beta_j x_{ij} \right)^2 \textbf{ subject to:}$$

$$\sum_{j=1}^{J} \beta_j^2 \leq t$$

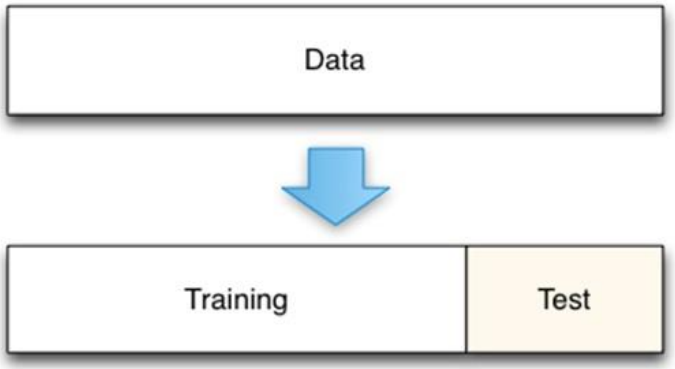
for decision variables α and β_j for $j = 1, \ldots, J$.

Ridge Regression

- Water resources, genetics, breast cancer prediction, forestry, and climate studies.
  - Multiple highly correlated features; it is not feasible to just drop features that are correlated with each other.
- Regression coefficient values are pushed towards zero, which reduces model complexity and enhances generalizability. The coefficients of non-predictive features will be small; the most predictive features will receive the most weight.
  - Effective even when collinearity is a problem.

Training & Testing

1. **Partitioning:** Data is split into non-overlapping partitions called the *training* and *testing* sets.
2. **Training Set:** This data set is used for model fitting, i.e., to build or train the model.
3. **Testing Set:** This data set is used for model validation, i.e., we evaluate the model on its out-of-sample prediction accuracy.



Training & Testing

- There are many different rules for setting the size of the training and testing sets.
  - 80% for training and 20% for validation.

- With more *training* data, your **parameter** estimates will be more precise.
- With more *testing* data, your **performance** statistics will be more precise.

Evaluating Regression Tasks

- For each element in the testing set, we compare the *prediction* made by our model to what the *actual* observation was.

Mean Squared Error (MSE)

$$MSE = \frac{1}{M}\sum_{j=1}^{M}\left(\hat{y}_j - y_j\right)^2$$

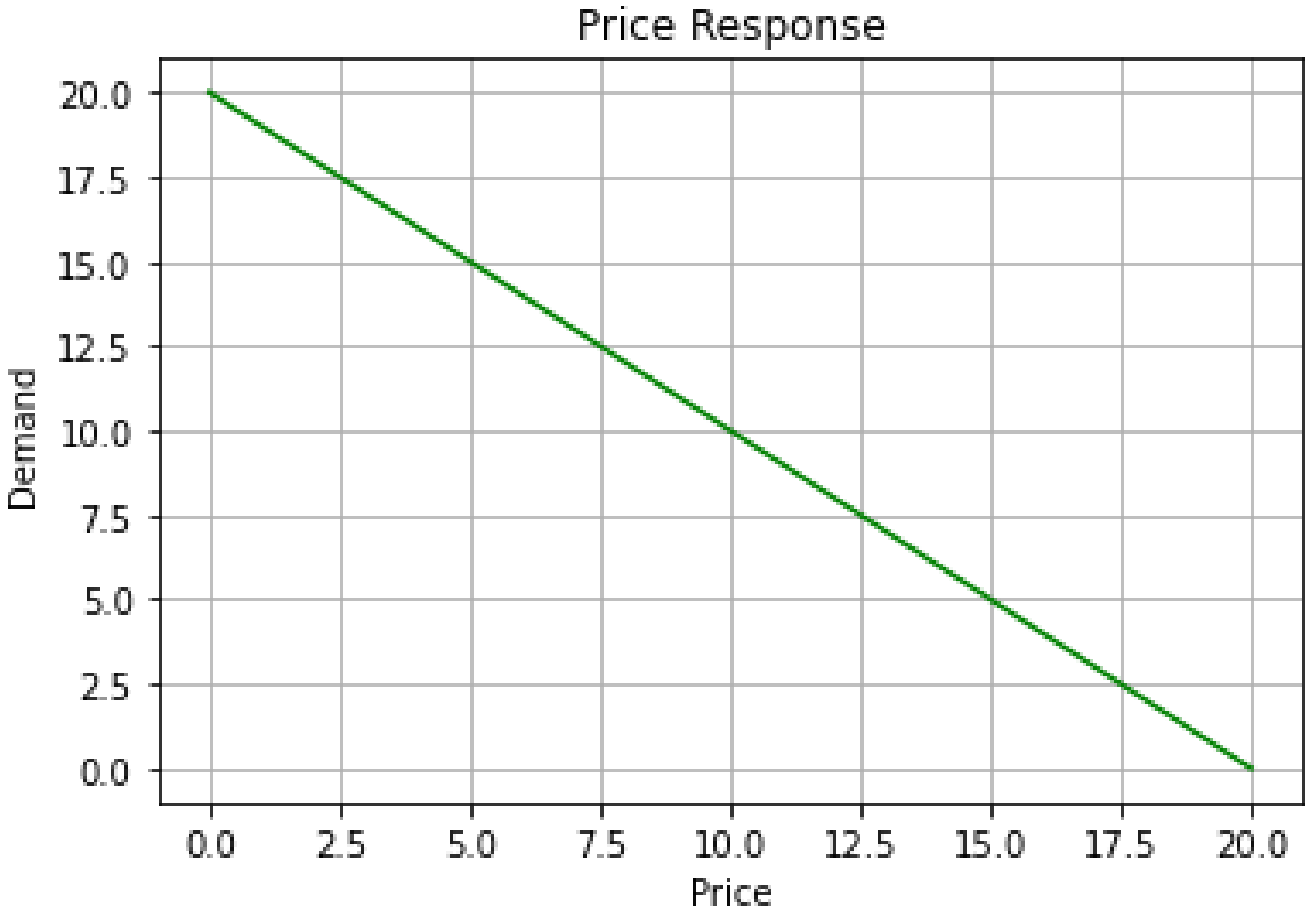
Performance metric gives higher penalty to large deviations from the actual observation. The result corresponds to the mean error.

Case Study: Student Performance

- **Summary:** A historical data set of the academic performance of high school students in Portugal.
- **Period:** September 1, 2012, to June 31, 2013.
- **Unit:** Each data point represents one student.
- **Features:** The school system, socioeconomic metrics, family size, historical performance, travel time, study time, mother's and father's education, internet connectivity, health, absenteeism, and aspirations.

The dataset contains many features. What are the most important when predicting grades?

Price Response Functions



Variable Pricing with Diversion

Variable Pricing with Diversion

Consider an industrial bakery that can bake up to 1,100 pallets of bread loaves per day. The cost of baking a pallet of bread is $19. Demand for different days of the week is independent and the forecasted demand curves are linear (see next slide). Further, a demand forecasting model predicts that for every $1 difference in price, 9 pallets of demand will be diverted from the day with the *higher* price to the day with the *lower* price. This means that, in general, demand will shift from days with higher-than average prices to days with lower-than-average prices. What is the optimal pricing strategy to maximize weekly profits if the maximum price of a pallet cannot exceed $40?

Variable Pricing with Diversion

Consider an industrial bakery that can bake up to 1,100 pallets of bread loaves per day. The demand forecasting model predicts that for every $1 difference in price, 9 pallets of demand will be diverted. The cost of baking a pallet of bread is $19. What is the pricing strategy that maximizes profit?

	Max Demand	Slope
Sunday	3100	62
Monday	1900	50
Tuesday	1700	40
Wednesday	1710	42
Thursday	2000	53
Friday	2500	54
Saturday	3300	60

Variable Pricing with Diversion

Price-Response Functions:

- Sunday: $d_1(p_1) = 3100 - 62p_1$
- Monday: $d_2(p_2) = 1900 - 50p_2$
- Tuesday: $d_3(p_3) = 1700 - 40p_3$
- Wednesday: $d_4(p_4) = 1710 - 42p_4$
- Thursday: $d_5(p_5) = 2000 - 53p_5$
- Friday: $d_6(p_6) = 2500 - 54p_6$
- Saturday: $d_7(p_7) = 3300 - 60p_7$

	Max Demand	Slope
Sunday	3100	62
Monday	1900	50
Tuesday	1700	40
Wednesday	1710	42
Thursday	2000	53
Friday	2500	54
Saturday	3300	60

Variable Pricing with Diversion

Define the objective:

Maximize the total profit

Decision variables:

Write the mathemtical objective function:

Variable Pricing with Diversion

Define the objective:

Maximize the total profit

Decision variables:

$\boldsymbol{p_n}$: the price of a pallet on day $\boldsymbol{n = 1, \ldots, 7.}$

$\boldsymbol{\widetilde{d}_n(p_n)}$: the demand for pallets on day $\boldsymbol{n = 1, \ldots, 7.}$

Write the mathemtical objective function:

Variable Pricing with Diversion

Define the objective:

Maximize the total profit

Decision variables:

$\boldsymbol{p_n}$: the price of a pallet on day $\boldsymbol{n = 1, \ldots, 7.}$

$\boldsymbol{\tilde{d}_n(p_n)}$: the demand for pallets on day $\boldsymbol{n = 1, \ldots, 7.}$

Write the mathemtical objective function:

$$\textbf{Maximize} \quad \mathbf{Z} = \sum_{n=1}^{7} (\boldsymbol{p_n} - \mathbf{19})\tilde{\boldsymbol{d}}_{\boldsymbol{n}}(\boldsymbol{p_n})$$

Variable Pricing with Diversion

Formulating the constraints

There are three types of constraints:

1. Demand constraints
2. Production constraints
3. Upper price constraints
4. Non-negativity constraints

Variable Pricing with Diversion

Formulating the demand constraints

Demand for bread on different days of the week follows the price-response function with one difference. For every $1 difference in price, 9 pallets of demand will be diverted.

$$\widetilde{\boldsymbol{d}}_{\boldsymbol{n}}(\boldsymbol{p}_{\boldsymbol{n}}) = \boldsymbol{d}_{\boldsymbol{n}}(\boldsymbol{p}_{\boldsymbol{n}}) + \qquad \boldsymbol{for\ n} = \mathbf{1}, \ldots, \mathbf{7}$$

Variable Pricing with Diversion

Formulating the demand constraints

Demand for bread on different days of the week follows the price-response function with one difference. For every $1 difference in price, 9 pallets of demand will be diverted.

$$\tilde{d}_n(p_n) = d_n(p_n) + 9 \sum_{m=1}^{7} (p_m - p_n) \quad for\ n = 1, \dots, 7$$

Variable Pricing with Diversion

Formulating the production constraints

Total demand should not exceed the number of pallets of bread that can be produced per day.

$$\tilde{d}_n(p_n) \leq 1100 \; for \; n = 1, \ldots, 7$$

Variable Pricing with Diversion

Formulating the upper price constraints

The price for a pallet of bread loaves on any day of the week must not exceed $40.

$$p_n \leq 40 \; for \; n = 1, \ldots, 7$$

Variable Pricing with Diversion

Maximize $\boldsymbol{Z} = \sum_{n=1}^{7}(p_n - 19)\tilde{d}_n(p_n)$

Subject to:

$\tilde{d}_n(p_n) = d_n(p_n) + 9\sum_{m=1}^{7}(p_m - p_n) \quad for\ n = 1,..,7$
(demand constraints)

$\tilde{d}_n(p_n) \leq 1100\ for\ n = 1, \ldots, 7$
(production constraints)

$p_n \leq 40\ for\ n = 1, \ldots, 7$
(upper price constraints)

$p_n \geq 0, \tilde{d}_n(p_n) \geq 0\ \ for\ n = 1, \ldots, 7$
(non-negativity constraints)

Variable Pricing with Diversion: Python Solution

- Differential pricing using variable strategies with diversion can extract additional profits.
- Using an *optimal* strategy ensures that **additional** customers can be served and smooths demand, i.e., the number of pallet loaves per day is equal.
  - Increased profits when compared with a static pricing.
- The difficulty in implementing this strategy is:
  - Forecasting the price-response curves.
  - Estimating the number of diverted customers.

What managerial intuition do you get from the Python solution?

Markdown Optimization

Markdown Optimization

It is currently the beginning of January and MEC® is looking to liquidate their current inventory of 160 winter snowsuits by the end of the selling season (which is 4-months away). Any unsold merchandise at this time will be considered unsellable and thus, will be donated to charity. To achieve this goal, MEC® wants to implement a *monthly markdown pricing strategy*. That is, in order to liquidate the remaining inventory, prices will be gradually reduced month-over-month from their highest point in January to their lowest point in April. A snowsuit consists of a jacket and a pair of snow pants. Note that snowsuits can be sold as a pair, or the jacket and the snow pants can be sold separately.

Markdown Optimization

Formulate and solve a nonlinear optimization model to determine the optimal monthly prices for the snowsuit, jacket, and snow pants in order to maximize total revenue.

	Max Demand (Snowsuit)	Max Demand (Jacket)	Max Demand (Snow Pants)	Slope (Snowsuit)	Slope (Jacket)	Slope (Snow Pants)
January (1)	80	120	50	0.5	0.7	0.8
February (2)	80	90	70	0.5	0.9	0.4
March (3)	30	80	40	0.5	1.0	0.4
April (4)	30	50	10	1.0	0.9	0.4

Markdown Optimization

Define the objective:

Maximize the total profit

Decision variables:

Write the mathematical objective function:

Markdown Optimization

Define the objective:

Maximize the total profit

Decision variables:

$\boldsymbol{p_{in}}$: price of product $\boldsymbol{i = 1, \ldots, 3}$ in month $\boldsymbol{n = 1, \ldots, 4.}$

$\boldsymbol{d_{in}(p_{in})}$: demand for product $\boldsymbol{i = 1, \ldots, 3}$ in month $\boldsymbol{n = 1, \ldots, 4.}$

Write the mathematical objective function:

Markdown Optimization

Define the objective:

Maximize the total profit

Decision variables:

p_{in} : price of product $i = 1, \ldots, 3$ in month $n = 1, \ldots, 4$.

$d_{in}(p_{in})$: demand for product $i = 1, \ldots, 3$ in month $n = 1, \ldots, 4$.

Write the mathematical objective function:

$$\text{Maximize } Z = \sum_{i=1}^{3} \sum_{n=1}^{4} p_{in} d_{in}(p_{in})$$

Markdown Optimization

Price-Response Functions (Snowsuit):

- January: $d_{11}(p_{11}) = 80 - 0.5p_{11}$
- February: $d_{12}(p_{12}) = 80 - 0.5p_{12}$
- March: $d_{13}(p_{13}) = 30 - 0.5p_{13}$
- April: $d_{14}(p_{14}) = 30 - 1.0p_{14}$

	Max Demand (Snowsuit)	Max Demand (Jacket)	Max Demand (Snow Pants)	Slope (Snowsuit)	Slope (Jacket)	Slope (Snow Pants)
January (1)	80	120	50	0.5	0.7	0.8
February (2)	80	90	70	0.5	0.9	0.4
March (3)	30	80	40	0.5	1.0	0.4
April (4)	30	50	10	1.0	0.9	0.4

Markdown Optimization

Price-Response Functions (Jacket):

- January: $d_{21}(p_{21}) = 120 - 0.7p_{21}$
- February: $d_{22}(p_{22}) = 90 - 0.9p_{22}$
- March: $d_{23}(p_{23}) = 80 - 1.0p_{23}$
- April: $d_{24}(p_{24}) = 50 - 0.9p_{24}$

	Max Demand (Snowsuit)	Max Demand (Jacket)	Max Demand (Snow Pants)	Slope (Snowsuit)	Slope (Jacket)	Slope (Snow Pants)
January (1)	80	120	50	0.5	0.7	0.8
February (2)	80	90	70	0.5	0.9	0.4
March (3)	30	80	40	0.5	1.0	0.4
April (4)	30	50	10	1.0	0.9	0.4

Markdown Optimization

Price-Response Functions (Snow Pants):

- January: $d_{31}(p_{31}) = 50 - 0.8p_{31}$
- February: $d_{32}(p_{32}) = 70 - 0.4p_{32}$
- March: $d_{33}(p_{33}) = 40 - 0.4p_{33}$
- April: $d_{34}(p_{34}) = 10 - 0.4p_{34}$

	Max Demand (Snowsuit)	Max Demand (Jacket)	Max Demand (Snow Pants)	Slope (Snowsuit)	Slope (Jacket)	Slope (Snow Pants)
January (1)	80	120	50	0.5	0.7	0.8
February (2)	80	90	70	0.5	0.9	0.4
March (3)	30	80	40	0.5	1.0	0.4
April (4)	30	50	10	1.0	0.9	0.4

Markdown Optimization

Formulating the constraints

There are three types of constraints:

1. Demand constraints
2. Pricing constraints
3. Non-negativity constraints
4. Integrality constraints

Markdown Optimization

Formulating the demand constraints

Total demand should not exceed the number of jackets and pants that are currently in inventory.

Markdown Optimization

Formulating the demand constraints

Total demand should not exceed the number of jackets and pants that are currently in inventory.

$$\sum_{n=1}^{4} \left(d_{1n}(p_{1n}) + d_{2n}(p_{2n}) \right) \leq 160$$

$$\sum_{n=1}^{4} \left(d_{1n}(p_{1n}) + d_{3n}(p_{3n}) \right) \leq 160$$

Markdown Optimization

Formulating the pricing constraints

The price in each month should be less than the previous month due to the Markdown strategy.

Markdown Optimization

Formulating the pricing constraints

The price in each month should be less than the previous month due to the Markdown strategy.

$$\boldsymbol{p_{in} \leq p_{i(n-1)}}$$

$$for\ all\ i = 1,2,3\ and\ n = 2,3,4$$

Markdown Optimization

Maximize $\boldsymbol{Z} = \sum_{i=1}^{3} \sum_{n=1}^{4} p_{in} d_{in}(p_{in})$

Subject to:

$\sum_{n=1}^{4} \left(d_{1n}(p_{1n}) + d_{2n}(p_{2n}) \right) \leq 160$
$\sum_{n=1}^{4} \left(d_{1n}(p_{1n}) + d_{3n}(p_{3n}) \right) \leq 160$
(demand constraints)

$p_{in} \leq p_{i(n-1)} \qquad for\ i = 1,2,3\ and\ n = 2,3,4$
(pricing constraints)

$p_{in} \geq 0, \;\; d_{in}(p_{in}) \geq 0 \;\; for\ i = 1,2,3\ and\ n = 1,2,3,4$
(non-negativity constraints)

$d_{in}(p_{in}) \in Integer \;\; for\ i = 1,2,3\ and\ n = 1,2,3,4$
(integrality constraints)

Markdown Optimization

- Differential pricing using markdown management strategies can extract additional profits.
  - From the seller's point of view, markdowns are useful when inventory is perishable (e.g., fashion, food, tech)
- Markdowns are **not** promotions or sales. A promotion is a *temporary* reduction in price while a markdown is a *permanent* reduction in price.
  - This is true for many industries (e.g., holiday items, home electronics, cars, clearances, bankruptcy).

What managerial intuition do you get from the Python solution?

Markdown Optimization: Python Solution

- In contrast to the last example, the decision variables associated with demand are **integer**.
  - **Reason 1:** Computational Runtime
    - If you try to run the Variable Pricing example using integer decision variables for demand, it will run for longer.
  - **Reason 2:** Managerial Interpretation
    - The number of units associated with jackets, pants, and snowsuits are integer-valued and the number is small. Thus, rounding may introduce large approximation errors.
  - **Reason 3:** Divisibility
    - It is possible to make fractional pallets of bread.

What managerial intuition do you get from the Python solution?

Next Class: Integer Programming Models

- The integer lattice, integer programming as an NP-hard problem, why rounding fractional solutions may not result in the optimal solution, linear relaxations, and the branch-and-bound technique.
- Scheduling, covering, and assignment problems.
- Expressing logical requirements as linear constraints with binary variables:
  - Conjunctive clauses ("*and*")
  - Disjunctive clauses ("*or*")
- Using auxiliary decision variables to express more complicated logical requirements and constraints.