OPTIMIZATION TECHNIQUES for DECISION MAKING

UNIT-2

Dr. Ravi Prakash Shahi

Professor in Analytics, AI, Machine Learning, Data Science, IoT, Software Engineering, Security, Computer Vision, Big Data

ravishahi71@gmail.com

LPP - INTEGER PROGRAMMING

- Integer Programming (IP) is a type of mathematical optimization where the objective function and constraints are linear, but the decision variables are restricted to take only integer values.
- This contrasts with LPP, where variables can take any real number value. Integer programming is essential for problems where decisions are inherently discrete, such as counting items, making yes/no choices, or scheduling tasks, logical decisions, and resource allocation.

What is Integer Programming?

In **Linear Programming**, variables can take **any real number** (e.g., 1.25, 3.7).
In **Integer Programming**, variables must be **whole numbers** (e.g., 1, 2, 3).

There are 3 main types:

1. **Pure Integer Programming** — All decision variables must be integers.
2. **Mixed Integer Programming (MIP)** — Some variables are integers; others can be real.
3. **Binary Integer Programming** — Variables can only be 0 or 1 (used for yes/no decisions).

Example — The Cargo Plane

- Imagine you’re managing logistics for a company. You have a small cargo plane with a weight capacity of **2,200 kg**. You have two types of high-value items you can ship.
- Your goal is to load the plane to maximize the total value of the cargo.

Your Items:

1. **Item A (Large Server Rack):**

   Value: **$2,000**

   Weight: **500 kg**

2. **Item B (Box of Microchips):**

   Value: **$1,200**

   Weight: **200 kg**

The critical rule is: **You cannot ship a fraction of an item.** You either ship a whole server rack or you don’t.

Step 1: Let's Try to Solve This with Linear Programming (The "Wrong" Way)

If we forget the "no fractions" rule for a moment, we can set this up as a standard LP problem.

- Let A be the number of Item A.
- Let B be the number of Item B.

Objective: Maximize Value = $2000A + 1200B$

Constraint: $500A + 200B \le 2200$

If we solve this LP (using the graphical method, checking the corners of the feasible region):

- One corner is shipping only Item A: 5A = 22 -> A = 4.4. The point is (4.4, 0).
- The other corner is shipping only Item B: 2B = 22 -> B = 11. The point is (0, 11).

  - At `(4.4, 0)` : Value = 4.4 * $2000 = **8,800**$.
  - At `(0, 11)` : Value = 11 * $1200 = **13,200**$.

The LP solution would be to ship **4.4 units of Item A** and 0 units of Item B.

This is the core problem: The best mathematical solution is physically impossible. You cannot ship 4.4 server racks.

Step 2: The Naive (and Often Wrong) Approach: Rounding

A common first thought is to just round the LP solution. Let's round (4.4, 0) down to the nearest integer plan: (4, 0).

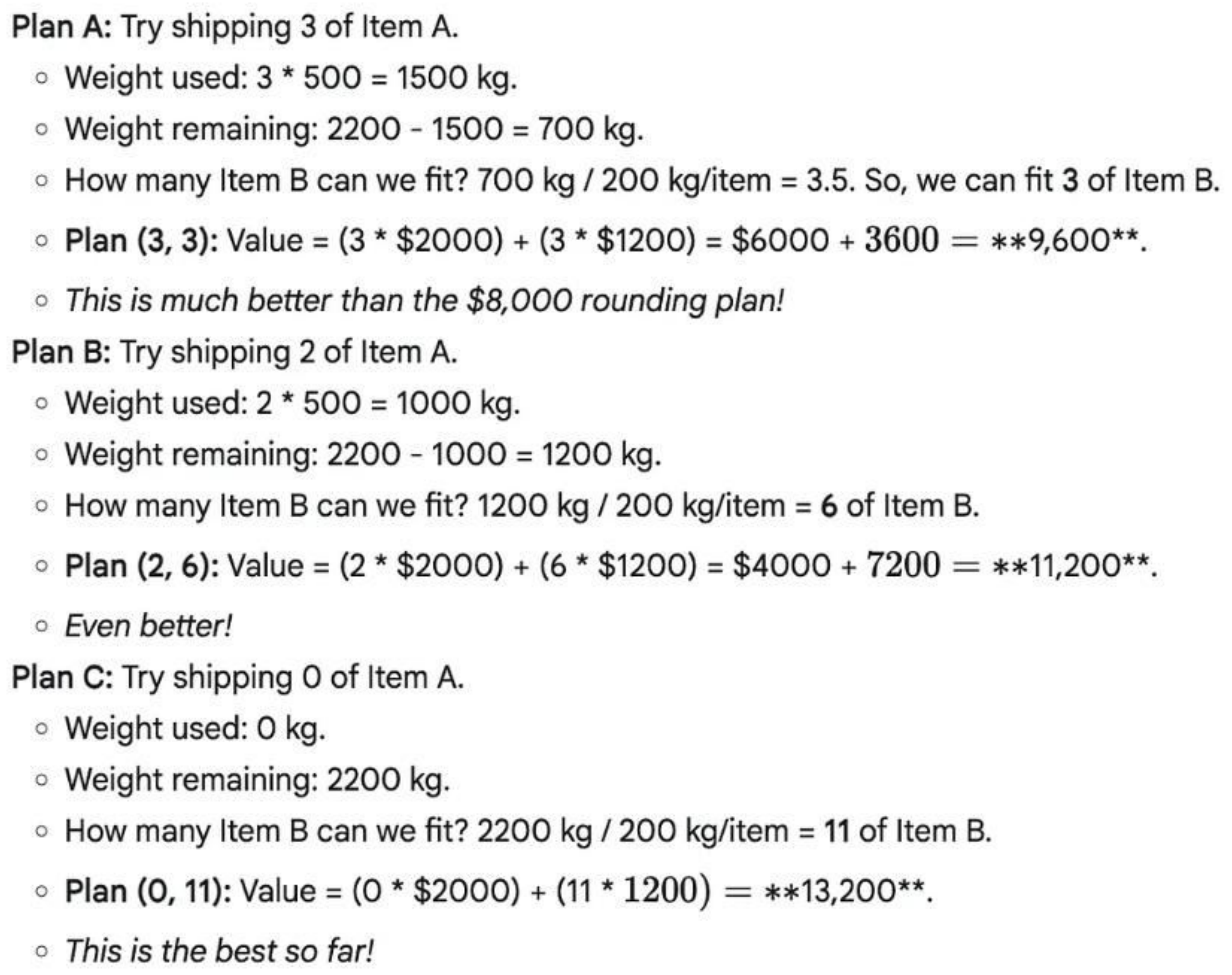
- **Plan (from rounding):** Ship 4 of Item A and 0 of Item B.
- **Weight:** 4 * 500 kg = 2000 kg (This is valid, as it's ≤ 2200 kg).
- Value: 4 * 2000 = 8000

Is this the best we can do? Maybe not.

Step 3: The Integer Programming Approach (Finding the True Best Plan)

Integer Programming finds the best solution among **only the whole-number possibilities.** The "map" of our feasible region is no longer a shaded area, but a set of individual, valid dots (the integer points).

Let’s check some other valid integer combinations to see if we can beat the $8,000 from rounding.



Plan A: Try shipping 3 of Item A.

- Weight used: 3 * 500 = 1500 kg.
- Weight remaining: 2200 - 1500 = 700 kg.
- How many Item B can we fit? 700 kg / 200 kg/item = 3.5. So, we can fit **3** of Item B.
- **Plan (3, 3):** Value = (3 * $2000) + (3 * $1200) = $6000 + 3600 = **9,600**.
- *This is much better than the $8,000 rounding plan!*

Plan B: Try shipping 2 of Item A.

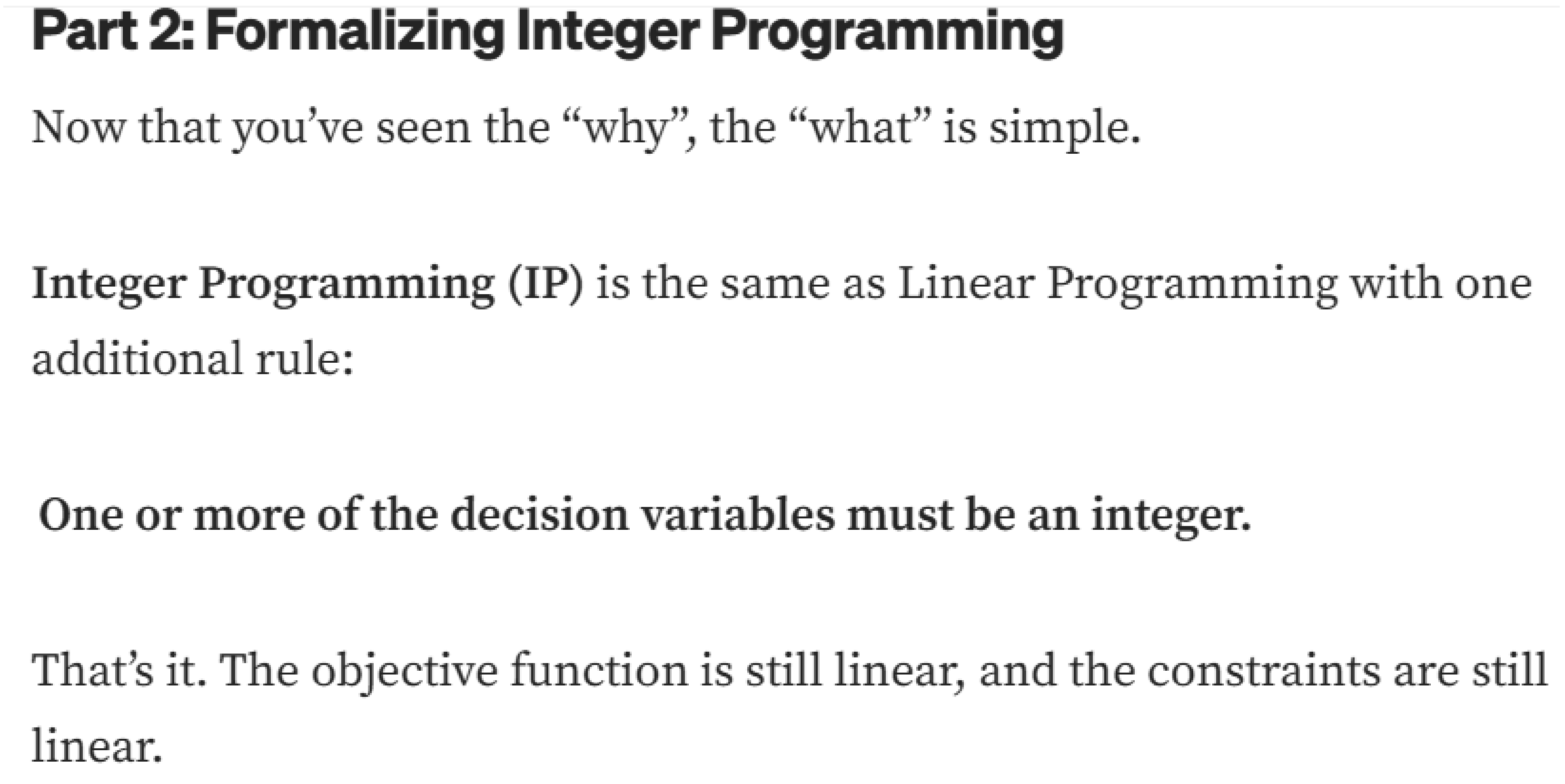
- Weight used: 2 * 500 = 1000 kg.
- Weight remaining: 2200 - 1000 = 1200 kg.
- How many Item B can we fit? 1200 kg / 200 kg/item = **6** of Item B.
- **Plan (2, 6):** Value = (2 * $2000) + (6 * $1200) = $4000 + 7200 = **11,200**.
- *Even better!*

Plan C: Try shipping 0 of Item A.

- Weight used: 0 kg.
- Weight remaining: 2200 kg.
- How many Item B can we fit? 2200 kg / 200 kg/item = **11** of Item B.
- **Plan (0, 11):** Value = (0 * $2000) + (11 * 1200) = **13,200**.
- *This is the best so far!*

By exploring the *integer* solutions, we find the true optimal plan.

Conclusion: The best possible plan is to ship **0 of Item A** and **11 of Item B** for a maximum value of **$13,200**. This solution is completely different from the one suggested by rounding the LP result.



Part 2: Formalizing Integer Programming

Now that you've seen the "why", the "what" is simple.

Integer Programming (IP) is the same as Linear Programming with one additional rule:

One or more of the decision variables must be an integer.

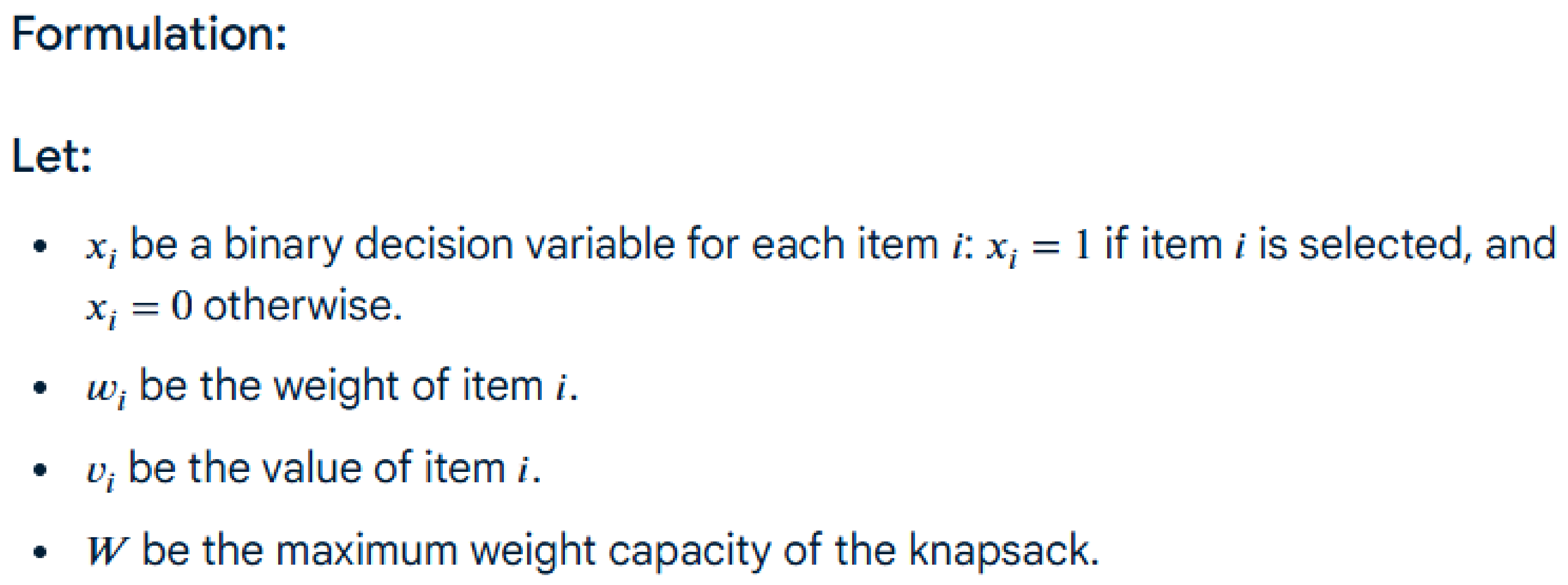
That's it. The objective function is still linear, and the constraints are still linear.

Types of Integer Programming

1. **Pure Integer Programming (PIP):**
   - **All** decision variables must be integers.
   - *Our cargo plane example was a PIP problem.*
2. **Mixed-Integer Programming (MIP or MILP):**
   - **Some** decision variables must be integers, while others are allowed to be fractional.
   - *Example:* A company deciding **how many new factories to build** (integer) and **how much to spend on marketing for each** (continuous/fractional).
3. **Binary Integer Programming (BIP) or 0/1 Integer Programming:**
   - A special, very common case where integer variables can only be **0 or 1**.
   - This is perfect for "Yes/No" decisions.
   - *Example:* A city deciding **which public projects to fund** (1 if funded, 0 if not), given a limited budget.

The Knapsack Problem

Consider a scenario where a hiker wants to maximize the value of items carried in a knapsack, subject to a weight limit. Each item has a specific weight and a specific value. The hiker cannot take fractions of items.



Formulation:

Let:

- x_i be a binary decision variable for each item i: $x_i = 1$ if item i is selected, and $x_i = 0$ otherwise.
- w_i be the weight of item i.
- v_i be the value of item i.
- W be the maximum weight capacity of the knapsack.

The Integer Program is then:

Maximize: $\sum_i v_i x_i$ (Total value of selected items)

Subject to:

- $\sum_i w_i x_i \leq W$ (Total weight must not exceed knapsack capacity)
- $x_i \in \{0, 1\}$ for all i (Each item is either selected or not selected)

Illustrative Data:

Suppose the knapsack capacity $W = 10$ kg, and there are three items:

- Item 1: Weight $w_1 = 4$ kg, Value $v_1 = 10$
- Item 2: Weight $w_2 = 6$ kg, Value $v_2 = 15$
- Item 3: Weight $w_3 = 3$ kg, Value $v_3 = 7$

Applying the IP:

The objective is to maximize $10x_1 + 15x_2 + 7x_3$ subject to $4x_1 + 6x_2 + 3x_3 \leq 10$, with $x_1, x_2, x_3 \in \{0, 1\}$.

Possible integer solutions (combinations of items) can be tested to find the optimal one that maximizes value without exceeding the weight limit. For instance, taking Item 1 and Item 3 ($x_1 = 1, x_3 = 1, x_2 = 0$) yields a total weight of $4 + 3 = 7$ kg and a total value of $10 + 7 = 17$. Taking only Item 2 ($x_2 = 1, x_1 = 0, x_3 = 0$) yields a total weight of 6 kg and a total value of 15. Taking Item 1 and Item 2 would exceed the weight limit ($4 + 6 = 10$ kg, but $10 + 15 = 25$ value). The optimal solution would be determined by a suitable algorithm for solving integer programs.

Example 1 – Workforce Scheduling Problem

- A hospital needs to schedule nurses for two shifts: Day and Night.
  - Each **Day shift** requires **5 nurses**.
  - Each **Night shift** requires **3 nurses**.
  - Only **8 nurses** are available in total.
- Decision variables:
  - x_1 = number of nurses assigned to Day shift
  - x_2 = number of nurses assigned to Night shift
- **Model:**

$$\text{Maximize } Z = 0 \quad \text{(feasibility problem)}$$

Subject to:

$$x_1 \geq 5, \quad x_2 \geq 3, \quad x_1 + x_2 \leq 8, \quad x_1, x_2 \in \mathbb{Z}^+$$

Interpretation: This ensures that the hospital assigns integer numbers of nurses to each shift while respecting staffing requirements.

NON-LINEAR PROGRAMMING

NON-LINEAR PROGRAMMING

- Nonlinear programming (NLP) is defined as a type of optimization problem where neither the objective function nor the constraints are required to be linear with respect to the decision variables, allowing for the representation of more complex relationships and models.
- It is a mathematical optimization technique used to find the optimal solution (maximum or minimum) of a non-linear objective function, subject to a set of constraints that can also be non-linear.
- Unlike linear programming, where all relationships are linear, NLP allows for more complex, realistic representations of real-world problems.

LINEAR PROGRAMMING VERSUS NONLINEAR PROGRAMMING

LINEAR PROGRAMMING	NONLINEAR PROGRAMMING
A method to achieve the best outcome in a mathematical model whose requirements are represented by linear relationships	A process of solving an optimization problem where the constraints or the objective functions are nonlinear
Helps to find the best solution to a problem using constraints that are linear	Helps to find the best solution to a problem using constraints that are nonlinear

Characteristics of Non-linear Programming

- **Non-linear objective function:** The function to be optimized (maximized or minimized) contains non-linear terms (e.g., quadratic, exponential, logarithmic).
- **Non-linear constraints:** The limitations or restrictions on the decision variables can also contain non-linear terms.
- **Potential for multiple local optima:** Unlike linear programs which guarantee a single global optimum if one exists, non-linear programs can have multiple local optima, making finding the global optimum more challenging.

Example: Product Mix with Volume Discounts

Consider a company producing two products, X and Y, with the goal of maximizing profit.

The profit function might be non-linear due to volume discounts on raw materials or production efficiencies that vary with output levels.

Objective Function:

Maximize $P = f(x, y)$ where P is the total profit, and x and y are the quantities of products X and Y produced. The function $f(x, y)$ could be non-linear, for instance: $P = (10x - 0.1x^2) + (15y - 0.2y^2)$ Here, the terms $0.1x^2$ and $0.2y^2$ represent decreasing marginal returns or increasing costs as production volume rises.

Constraints:

The production is subject to constraints on resources like labor hours and machine capacity. These constraints could also be non-linear. For example, if the efficiency of labor or machines changes with the scale of production: $2x^2 + 3y \leq 100$ (Labor hours constraint) $x + y^2 \leq 50$ (Machine capacity constraint) $x, y \geq 0$ (Non-negativity constraints)

In this example, the non-linear terms in both the objective function and the constraints necessitate the use of non-linear programming techniques to find the optimal production quantities of X and Y that maximize profit. Solving such problems often involves iterative numerical methods like gradient descent or specialized algorithms designed for non-linear optimization.

QUADRATIC PROGRAMMING

- Quadratic Programming (QP) is an optimization method used to find the minimum or maximum of a quadratic objective function, subject to linear equality and inequality constraints.
- It is a specific type of nonlinear programming where the objective function is quadratic, and all constraints are linear.

Big Picture

1. QP is a special case of nonlinear programming
2. The objective function is quadratic, a second order polynomial of decision variables
3. Global minimum exists if the quadratic form is positive definite (or the function is strictly convex)
4. There may be constraints, which may or may not be binding

The general form of a Quadratic Programming problem is:

Minimize: $f(x) = \frac{1}{2} x^T H x + c^T x$

Subject to:

$A_{eq}x = b_{eq}$ (Linear equality constraints) $A_{ineq}x \leq b_{ineq}$ (Linear inequality constraints) $l \leq x \leq u$ (Bound constraints on variables)

Where:

- x is the vector of decision variables.
- H is a symmetric matrix (Hessian matrix) representing the quadratic terms.
- c is a vector representing the linear terms.
- $A_{eq'}$ $b_{eq'}$ $A_{ineq'}$ $b_{ineq'}$ l, and u are matrices and vectors defining the linear constraints and variable bounds.

Example: Portfolio Optimization

Consider a simplified portfolio optimization problem where an investor wants to minimize the risk (variance) of a portfolio consisting of two assets, while achieving a certain expected return and respecting budget constraints.

Let x_1 and x_2 be the proportions of wealth invested in asset 1 and asset 2, respectively.

Objective Function (Minimize Risk):

Assume the portfolio variance is given by: $f(x_1, x_2) = 0.5 \cdot (2x_1^2 + 3x_2^2 + 0.5x_1x_2)$

Constraints:

- **Budget Constraint (Equality):** The sum of proportions must be 1.

$x_1 + x_2 = 1$

- **Minimum Expected Return (Inequality):** The expected return must be at least 0.15.

Assume expected returns are 0.1 for asset 1 and 0.2 for asset 2. $0.1x_1 + 0.2x_2 \geq 0.15$

- **Non-negativity Constraints (Bounds):** Proportions cannot be negative.

$x_1 \geq 0$ $x_2 \geq 0$

This problem fits the Quadratic Programming framework because the objective function is quadratic, and all constraints are linear. Solving this QP problem would yield the optimal proportions x_1 and x_2 that minimize portfolio risk while satisfying the return and budget requirements.

OPTIMIZATION TECHNIQUES for DECISION MAKING

UNIT-2

Dr. Ravi Prakash Shahi

Professor in Analytics, AI, Machine Learning, Data Science, IoT, Software Engineering, Security, Computer Vision, Big Data

ravishahi71@gmail.com

LPP - INTEGER PROGRAMMING

- Integer Programming (IP) is a type of mathematical optimization where the objective function and constraints are linear, but the decision variables are restricted to take only integer values.
- This contrasts with LPP, where variables can take any real number value. Integer programming is essential for problems where decisions are inherently discrete, such as counting items, making yes/no choices, or scheduling tasks, logical decisions, and resource allocation.

What is Integer Programming?

In **Linear Programming**, variables can take **any real number** (e.g., 1.25, 3.7).
In **Integer Programming**, variables must be **whole numbers** (e.g., 1, 2, 3).

There are 3 main types:

1. **Pure Integer Programming** — All decision variables must be integers.
2. **Mixed Integer Programming (MIP)** — Some variables are integers; others can be real.
3. **Binary Integer Programming** — Variables can only be 0 or 1 (used for yes/no decisions).

Example — The Cargo Plane

- Imagine you’re managing logistics for a company. You have a small cargo plane with a weight capacity of **2,200 kg**. You have two types of high-value items you can ship.
- Your goal is to load the plane to maximize the total value of the cargo.

Your Items:

1. **Item A (Large Server Rack):**

   Value: **$2,000**

   Weight: **500 kg**

2. **Item B (Box of Microchips):**

   Value: **$1,200**

   Weight: **200 kg**

The critical rule is: **You cannot ship a fraction of an item.** You either ship a whole server rack or you don’t.

Step 1: Let's Try to Solve This with Linear Programming (The "Wrong" Way)

If we forget the "no fractions" rule for a moment, we can set this up as a standard LP problem.

- Let A be the number of Item A.
- Let B be the number of Item B.

Objective: Maximize Value = $2000A + 1200B$

Constraint: $500A + 200B \leq 2200$

If we solve this LP (using the graphical method, checking the corners of the feasible region):

- One corner is shipping only Item A: 5A = 22 -> A = 4.4. The point is (4.4, 0).
- The other corner is shipping only Item B: 2B = 22 -> B = 11. The point is (0, 11).

  - At `(4.4, 0)` : Value = 4.4 * 2000 = **8,800**.
  - At `(0, 11)` : Value = 11 * 1200 = **13,200**.

The LP solution would be to ship **4.4 units of Item A** and 0 units of Item B.

This is the core problem: The best mathematical solution is physically impossible. You cannot ship 4.4 server racks.

Step 2: The Naive (and Often Wrong) Approach: Rounding

A common first thought is to just round the LP solution. Let's round (4.4, 0) down to the nearest integer plan: (4, 0).

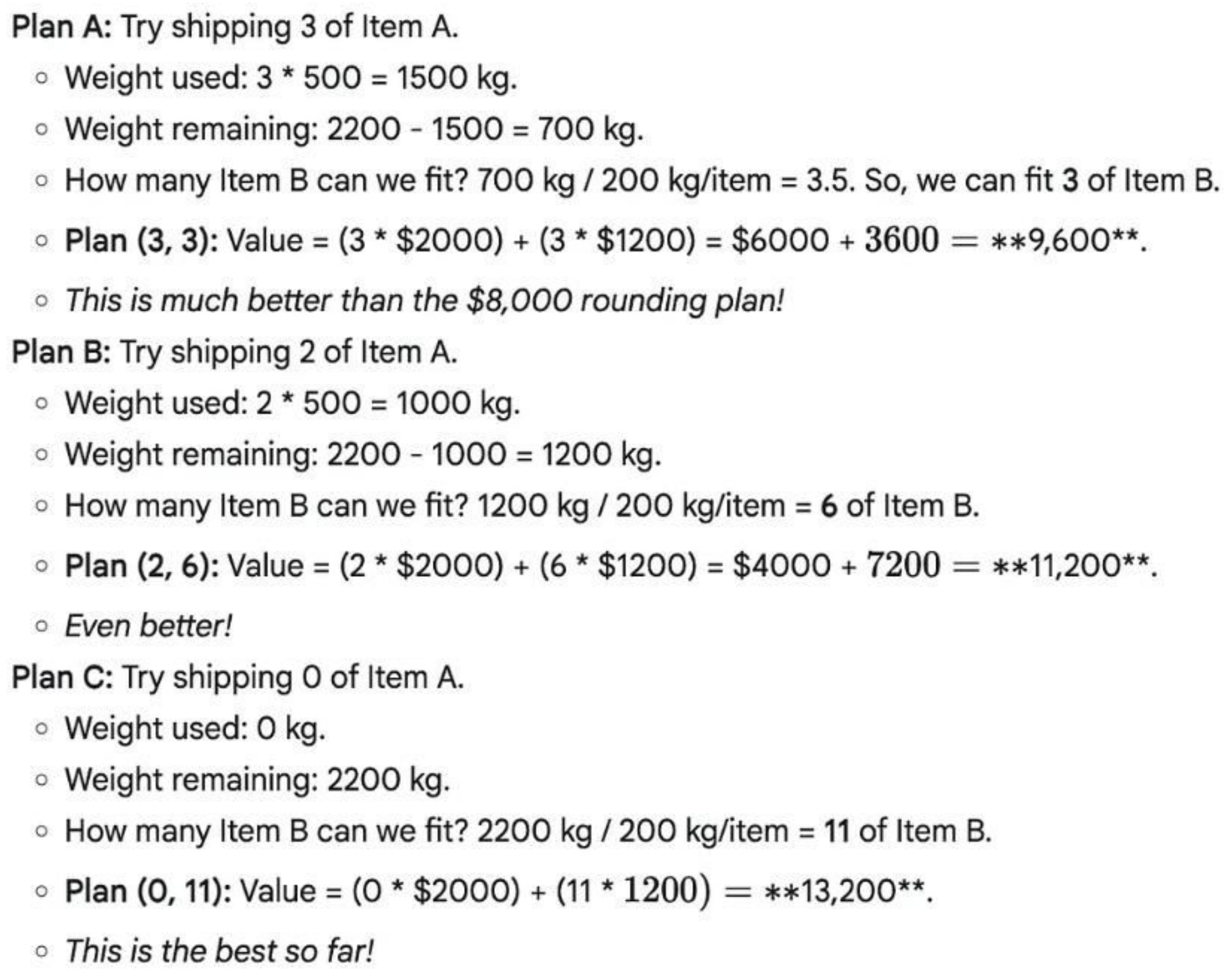
- **Plan (from rounding):** Ship 4 of Item A and 0 of Item B.
- **Weight:** 4 * 500 kg = 2000 kg (This is valid, as it's ≤ 2200 kg).
- Value: 4 * 2000 = 8000

Is this the best we can do? Maybe not.

Step 3: The Integer Programming Approach (Finding the True Best Plan)

Integer Programming finds the best solution among **only the whole-number possibilities.** The "map" of our feasible region is no longer a shaded area, but a set of individual, valid dots (the integer points).

Let’s check some other valid integer combinations to see if we can beat the $8,000 from rounding.



Plan A: Try shipping 3 of Item A.

- Weight used: 3 * 500 = 1500 kg.
- Weight remaining: 2200 - 1500 = 700 kg.
- How many Item B can we fit? 700 kg / 200 kg/item = 3.5. So, we can fit **3** of Item B.
- **Plan (3, 3):** Value = (3 * $2000) + (3 * $1200) = $6000 + 3600 = **9,600**.
- *This is much better than the $8,000 rounding plan!*

Plan B: Try shipping 2 of Item A.

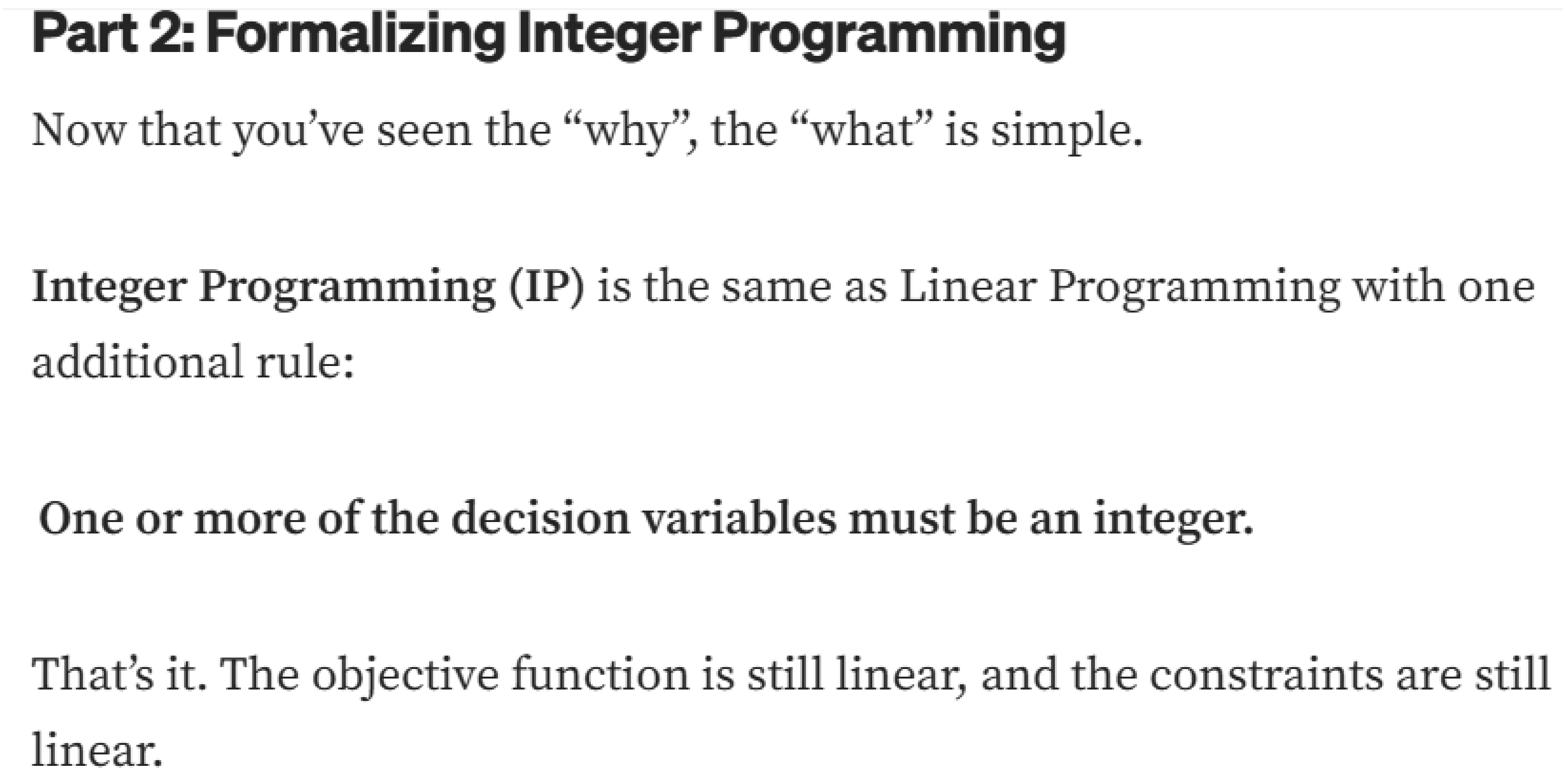
- Weight used: 2 * 500 = 1000 kg.
- Weight remaining: 2200 - 1000 = 1200 kg.
- How many Item B can we fit? 1200 kg / 200 kg/item = **6** of Item B.
- **Plan (2, 6):** Value = (2 * $2000) + (6 * $1200) = $4000 + 7200 = **11,200**.
- *Even better!*

Plan C: Try shipping 0 of Item A.

- Weight used: 0 kg.
- Weight remaining: 2200 kg.
- How many Item B can we fit? 2200 kg / 200 kg/item = **11** of Item B.
- **Plan (0, 11):** Value = (0 * $2000) + (11 * 1200) = **13,200**.
- *This is the best so far!*

By exploring the *integer* solutions, we find the true optimal plan.

Conclusion: The best possible plan is to ship **0 of Item A** and **11 of Item B** for a maximum value of **$13,200**. This solution is completely different from the one suggested by rounding the LP result.



Part 2: Formalizing Integer Programming

Now that you've seen the "why", the "what" is simple.

Integer Programming (IP) is the same as Linear Programming with one additional rule:

One or more of the decision variables must be an integer.

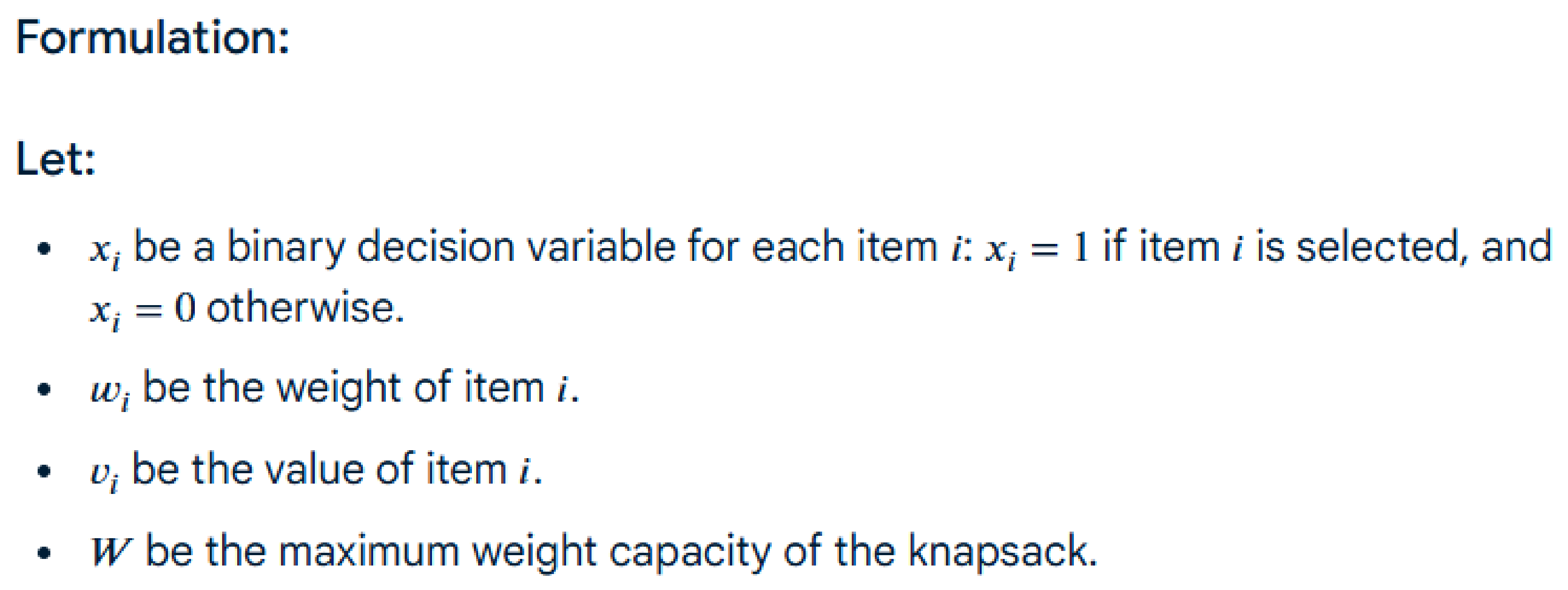
That's it. The objective function is still linear, and the constraints are still linear.

Types of Integer Programming

1. **Pure Integer Programming (PIP):**
   - **All** decision variables must be integers.
   - *Our cargo plane example was a PIP problem.*
2. **Mixed-Integer Programming (MIP or MILP):**
   - **Some** decision variables must be integers, while others are allowed to be fractional.
   - *Example:* A company deciding **how many new factories to build** (integer) and **how much to spend on marketing for each** (continuous/fractional).
3. **Binary Integer Programming (BIP) or 0/1 Integer Programming:**
   - A special, very common case where integer variables can only be **0 or 1**.
   - This is perfect for "Yes/No" decisions.
   - *Example:* A city deciding **which public projects to fund** (1 if funded, 0 if not), given a limited budget.

The Knapsack Problem

Consider a scenario where a hiker wants to maximize the value of items carried in a knapsack, subject to a weight limit. Each item has a specific weight and a specific value. The hiker cannot take fractions of items.



Formulation:

Let:

- x_i be a binary decision variable for each item i: $x_i = 1$ if item i is selected, and $x_i = 0$ otherwise.
- w_i be the weight of item i.
- v_i be the value of item i.
- W be the maximum weight capacity of the knapsack.

The Integer Program is then:

Maximize: $\sum_i v_i x_i$ (Total value of selected items)

Subject to:

- $\sum_i w_i x_i \leq W$ (Total weight must not exceed knapsack capacity)
- $x_i \in \{0, 1\}$ for all i (Each item is either selected or not selected)

Illustrative Data:

Suppose the knapsack capacity $W = 10$ kg, and there are three items:

- Item 1: Weight $w_1 = 4$ kg, Value $v_1 = 10$
- Item 2: Weight $w_2 = 6$ kg, Value $v_2 = 15$
- Item 3: Weight $w_3 = 3$ kg, Value $v_3 = 7$

Applying the IP:

The objective is to maximize $10x_1 + 15x_2 + 7x_3$ subject to $4x_1 + 6x_2 + 3x_3 \leq 10$, with $x_1, x_2, x_3 \in \{0, 1\}$.

Possible integer solutions (combinations of items) can be tested to find the optimal one that maximizes value without exceeding the weight limit. For instance, taking Item 1 and Item 3 ($x_1 = 1, x_3 = 1, x_2 = 0$) yields a total weight of $4 + 3 = 7$ kg and a total value of $10 + 7 = 17$. Taking only Item 2 ($x_2 = 1, x_1 = 0, x_3 = 0$) yields a total weight of 6 kg and a total value of 15. Taking Item 1 and Item 2 would exceed the weight limit ($4 + 6 = 10$ kg, but $10 + 15 = 25$ value). The optimal solution would be determined by a suitable algorithm for solving integer programs.

Example 1 – Workforce Scheduling Problem

- A hospital needs to schedule nurses for two shifts: Day and Night.
  - Each **Day shift** requires **5 nurses**.
  - Each **Night shift** requires **3 nurses**.
  - Only **8 nurses** are available in total.
- Decision variables:
  - x_1 = number of nurses assigned to Day shift
  - x_2 = number of nurses assigned to Night shift
- **Model:**

$$\text{Maximize } Z = 0 \quad \text{(feasibility problem)}$$

Subject to:

$$x_1 \geq 5, \quad x_2 \geq 3, \quad x_1 + x_2 \leq 8, \quad x_1, x_2 \in \mathbb{Z}^+$$

Interpretation: This ensures that the hospital assigns integer numbers of nurses to each shift while respecting staffing requirements.