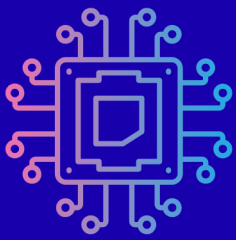
2026 Edition

The Future Coder Series

Computer Science စာအုပ်

Vol 3: The Math Of Computing



ရန်နိုင်လင်း (Software Engineer)

MBA, PG Dip (CS), BSc(Physics)

Chapter 1: Matrices & Data

1.1 Matrix (မက်ထရစ်) ဆိုတာ ဘာလဲ?

1.2 ဓာတ်ပုံများနှင့် Matrix (Modern Application)

1.3 Matrix တွက်ချက်နည်းများ (Basic Operations)

1.4 Vector (ဗက်တာ) - လမ်းညွှန်ပြ မြှားတံများ

Chapter 2: Systems of Linear Equations

2.1 System of Equations ဆိုတာ ဘာလဲ?

2.2 မျဉ်းကြောင်းများဖြင့် မြင်ယောင်ကြည့်ခြင်း (Visualizing Solutions)

2.3 Computer Graphics & Games (ဂိမ်းထဲက 3D ပုံရိပ်များ)

Chapter 3: Functions & Numerical Methods

3.1 Function (ဖန်ရှင်) ဆိုတာ ဘာလဲ?

3.2 Numerical Methods (ဂဏန်းဖြင့် ခန့်မှန်းနည်းများ)

3.3 Modern Applications (လက်တွေ့ ကမ္ဘာအသုံးချမှု)

Chapter 4: Linear Programming

4.1 Linear Programming ဆိုတာ ဘာလဲ?

4.2 လက်တွေ့ ဥပမာ (ပရိဘောဂ စက်ရုံ)

4.3 ကွန်ပျူတာ နားလည်အောင် ဘာသာပြန်ခြင်း

Chapter 5: Statistics & Probability

Chapter 1: Matrices & Data

(မက်ထရစ်များနှင့် ဒေတာသိပ္ပံ အခြေခံ)

1.1 Matrix (မက်ထရစ်) ဆိုတာ ဘာလဲ?

ကွန်ပျူတာမျက်နှာပြင် (Screen) ပေါ်က အရာအားလုံးဟာ ဂဏန်းတွေ စီထားတဲ့ ဇယားကွက်တွေ ဖြစ်ပါတယ်။ အဲဒီ ဂဏန်းဇယားကွက်ကို သချိ်ာလို **Matrix** လို့ ခေါ်ပါတယ်။

ကျောင်းမှာ သင်ဖူးတဲ့ ဇယား (Table) တွေနဲ့ တူတူပါပဲ။

ဥပမာ - မိုးလေဝသ မှတ်တမ်း :

ရန်ကုန်၊ မန္တလေး၊ မော်လမြိုင် မြို့တွေရဲ့ အပူချိန်ကို ဇယားနဲ့ ရေးကြည့်မယ်။

မြို့	အမြင့်ဆုံး (High)	အနိမ့်ဆုံး (Low)
ရန်ကုန်	94	63
မန္တလေး	96	73
မော်လမြိုင်	81	60

ဒါကို သချိ်ာ (Matrix) ပုံစံ ပြောင်းရေးရင် ဒီလိုရပါတယ်:

$$\begin{bmatrix} 94 & 63 \\ 96 & 73 \\ 81 & 60 \end{bmatrix}$$

ဒါကို **3 x 2 Matrix** (အတန်း ၃ တန်း၊ တိုင် ၂ တိုင် ရှိသော မက်ထရစ်) လို့ ခေါ်ပါတယ် ။

1.2 ဓာတ်ပုံများနှင့် Matrix (Modern Application)

ကွန်ပျူတာ၊ ဖုန်းထဲက ဓာတ်ပုံတစ်ပုံဟာ တကယ်တော့ အရောင်အနုအရင့်ကို ကိုယ်စားပြုတဲ့ ဂဏန်းတွေ (Matrix) သာ ဖြစ်ပါတယ်။

(a) Black & White Image (အဖြူအမည်း ပုံ)

ပုံတစ်ပုံကို အကွက်စိတ်လေးတွေ (Pixels) ချလိုက်ရင် -

- 0 = အမည်းရောင် (Black)
- 255 = အဖြူရောင် (White)
- ကြားထဲက ဂဏန်းတွေက မီးခိုးရောင် (Grey) ဖြစ်ပါတယ်။

ကွန်ပျူတာက သင့်ရဲ့ Selfie ပုံကို မြင်တဲ့အခါ မျက်နှာလို့ မမြင်ပါဘူး။ [120, 200, 55...] ဆိုတဲ့ Matrix ကြီးတစ်ခုအနေနဲ့ပဲ မြင်ပါတယ်။ ဒါကြောင့် AI တွေက လူမျက်နှာကို မှတ်မိတာဟာ ဒီဂဏန်းတွေကို တွက်ချက်လိုက်လို့ ဖြစ်ပါတယ်။

1.3 Matrix တွက်ချက်နည်းများ (Basic Operations)

ဂဏန်းတွေကို ပေါင်းသလိုပဲ၊ Matrix တွေကိုလည်း ပေါင်းလို့ မြှောက်လို့ ရပါတယ်။ ဒါပေမဲ့ သူ့မှာ သူ့သတ်မှတ်ချက်တွေ ရှိပါတယ်။

(a) Matrix Addition (ပေါင်းခြင်း)

- **သဘောတရား:** အရွယ်အစားတူတဲ့ ဇယားနှစ်ခုကို ထပ်ပြီး ပေါင်းလိုက်တာပါ။
- **လက်တွေ့ ဥပမာ:** "ဇန်နဝါရီလ ရောင်းရငွေ" ဇယားနဲ့ "ဖေဖော်ဝါရီလ ရောင်းရငွေ" ဇယားကို ပေါင်းလိုက်ရင် "နှစ်လစာ စုစုပေါင်း" ရလာသလိုပါပဲ။

$$\begin{bmatrix} 100 & 200 \\ 150 & 50 \end{bmatrix} + \begin{bmatrix} 20 & 50 \\ 30 & 10 \end{bmatrix} = \begin{bmatrix} 120 & 250 \\ 180 & 60 \end{bmatrix}$$

(b) Scalar Multiplication (ကိန်းသေနှင့် မြှောက်ခြင်း)

- **သဘောတရား:** Matrix ထဲက ဂဏန်းအားလုံးကို တန်ဖိုးတစ်ခုနဲ့ လိုက် မြှောက်တာပါ။
- **လက်တွေ့ ဥပမာ (Photo Editing):**
  - ဓာတ်ပုံတစ်ပုံ အရမ်းမှောင်နေတယ် ဆိုပါစို့။
  - အဲ့ဒီပုံ (Matrix) ထဲက ဂဏန်းအားလုံးကို 2 နဲ့ မြှောက်လိုက်ရင် (Double Brightness)၊ အရောင်တန်ဖိုးတွေ များလာပြီး ပုံက **လင်း** သွား ပါလိမ့်မယ်။

$$2 \times \begin{bmatrix} 10 & 50 \\ 20 & 40 \end{bmatrix} = \begin{bmatrix} 20 & 100 \\ 40 & 80 \end{bmatrix}$$

(c) Matrix Multiplication (မက်ထရစ်ချင်း မြှောက်ခြင်း)

- **သဘောတရား:** ဒါက နည်းနည်းဆန်းပါတယ်။ ပထမဇယား က အတန်း (Row) နဲ့ ဒုတိယဇယားက တိုင် (Column) ကို တွဲမြှောက်ရ တာပါ။
- **လက်တွေ့ ဥပမာ (ဈေးဝယ်ခြင်း):**
  - **Table A (ဈေးနှုန်း):** ကြက်သွန် ၁ ပိဿာ ၅ ကျပ်၊ အာလူး ၁ ပိဿာ ၆ ကျပ် [5]။
  - **Table B (ဝယ်ယူမှု):** ပထမလမှာ ကြက်သွန် ၃ ပိဿာ၊ အာလူး ၁ ပိဿာ ဝယ်တယ်။
  - **Result:** (3 x 5) + (1 x 6) = 21 ကျပ် ကျသင့်မယ်။
  - ဒီလိုမျိုး Data တွေအများကြီး (ဥပမာ - တစ်နိုင်ငံလုံးက ဈေးဝယ် စာရင်း) ကို တွက်ချက်ချင်ရင် Matrix Multiplication ကို သုံးရပါတယ်။

ဈေးနှုန်းဇယား

ပစ္စည်း	ဈေးနှုန်း (ကျပ်)
ကြက်သွန်	5
အာလူး	6

ဝယ်ယူမှုစာရင်း

အမျိုးအစား	ကြက်သွန် (ပိဿာ)	အာလူး (ပိဿာ)
အရေအတွက်	3	1

ရလဒ်

$$[3 \quad 1] \times \begin{bmatrix} 5 \\ 6 \end{bmatrix} = \begin{bmatrix} (3 \times 5) + (1 \times 6) \end{bmatrix} = [21]$$

1.4 Vector (ဗက်တာ) - လမ်းညွှန်ပြ မြှားတံများ

Matrix မှာ အတန်းတစ်တန်းတည်း (သို့) တိုင်တစ်တိုင်တည်း ရှိနေရင် အဲ့ဒါကို **Vector** လို့ခေါ်ပါတယ်။

Game Development ဥပမာ:

- မင်းဆော့နေတဲ့ ဂိမ်းထဲက ဇာတ်ကောင် (Character) ရဲ့ နေရာကို Vector နဲ့ သတ်မှတ်ပါတယ်။

- Position = [x, y] (ဥပမာ - [10, 5] ဆိုရင် ညာဘက် ၁၀ နေရာ၊ အပေါ် ၅ နေရာမှာ ရှိနေတယ်)။
- ဇာတ်ကောင်ကို ရွေ့တိုးချင်ရင် အဲ့ဒီ Vector ကို ဂဏန်းပေါင်းထည့်လိုက်ရုံပါပဲ။

Chapter 2: Systems of Linear Equations

(ညီမျှခြင်းများနှင့် ပြဿနာဖြေရှင်းနည်းများ)

2.1 System of Equations ဆိုတာ ဘာလဲ?

"ညီမျှခြင်း စနစ်" (System of Equations) ဆိုတာ ညီမျှခြင်းတစ်ခုတည်းနဲ့ အဖြေရှာလို့မရတဲ့အခါ၊ ညီမျှခြင်း (၂) ခု သို့မဟုတ် (၂) ခုထက်ပိုပြီး တွဲဖက်စဉ်းစားတာကို ခေါ်ပါတယ်။

ဥပမာ (စွေးဝယ်ပုစ္ဆာ):

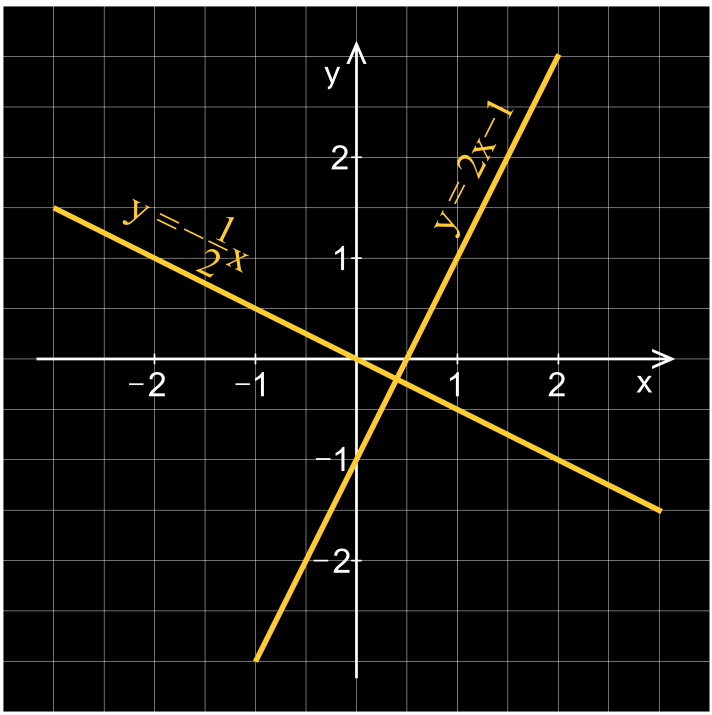
- သင်က **ပန်းသီး ၂ လုံး** နဲ့ **လိမ္မော်သီး ၁ လုံး** ဝယ်တော့ **၁၀ ကျပ်** ကျတယ်။
- **ပန်းသီး ၁ လုံး** နဲ့ **လိမ္မော်သီး ၁ လုံး** ဝယ်တော့ **၆ ကျပ်** ကျတယ်။
- ဒါဆို ပန်းသီး တစ်လုံး ဘယ်လောက်လဲ? လိမ္မော်သီး တစ်လုံး ဘယ်လောက်လဲ?

ဒါကို ကွန်ပျူတာ (သို့မဟုတ် သခ်ၤာ) ဘာသာစကားနဲ့ ရေးရင်:

1. 2x + 1y = 10

2. 1x + 1y = 6

ဒီညီမျှခြင်း နှစ်ခုကို တွဲဖြေလိုက်ရင် x=4 (ပန်းသီး), y=2 (လိမ္မော်သီး) ဆို တဲ့ အဖြေထွက်လာပါတယ်။ ဒါကို **System of Linear Equations** ဖြေရှင်းတယ် လို့ ခေါ်ပါတယ်။



2.2 မျဉ်းကြောင်းများဖြင့် မြင်ယောင်ကြည့်ခြင်း (Visualizing Solutions)

Linear Equation တစ်ခုစီဟာ ဂရပ် (Graph) ပေါ်မှာဆွဲလိုက်ရင် **မျဉ်းဖြောင့် (Line)** တစ်ခု ဖြစ်ပါတယ်။

- **အဖြေရှာခြင်း (Solution):** ညီမျှခြင်းနှစ်ခုရဲ့ အဖြေဆိုတာ အဲ့ဒီ မျဉ်းဖြောင့်နှစ်ခု **ဖြတ်သွားတဲ့ နေရာ (Intersection Point)** ပါပဲ။
- **အဖြေမရှိခြင်း (Inconsistent):** အကယ်၍ မျဉ်းနှစ်ကြောင်းက "ရထားသံလမ်း" လို ပြိုင်နေမယ် (Parallel) ဆိုရင် ဘယ်တော့မှ မဆုံပါဘူး။ ဒါဆိုရင် အဖြေမရှိပါဘူး (No Solution)။

2.3 Computer Graphics & Games (ဂိမ်းထဲက 3D ပုံရိပ်များ)

မင်းဆော့နေတဲ့ ဂိမ်းထဲက ဇာတ်ကောင်တွေ လှုပ်ရှားနေတာ၊ သေနတ်ပစ်လိုက်လို့ ကျည်ဆန်ထွက်သွားတာတွေဟာ Linear Equations တွေကို တစ်စက္ကန့် အတွင်း အကြိမ်ပေါင်းများစွာ ဖြေရှင်းနေတာ ဖြစ်ပါတယ်။

- **Movement (ရွှေ့လျားခြင်း):** ဇာတ်ကောင်က ရှေ့ကို ခြေတစ်လှမ်းတိုးလိုက်တာဟာ သူ့ရဲ့ တည်နေရာ (x, y, z) ကို ညီမျှခြင်းတစ်ခုနဲ့ ပေါင်းထည့်လိုက်တာ ဖြစ်ပါတယ်။
- **Collision Detection (တိုက်မိခြင်း):** ကျည်ဆန်က ရန်သူကို ထိလား မထိလား သိဖို့အတွက်၊ ကျည်ဆန်သွား
  တဲ့ လမ်းကြောင်း (Line Equation) နဲ့ ရန်သူရှိတဲ့ နေရာ ဆုံမှတ် ရှိမရှိ တွက်ချက်ရပါတယ်။

Chapter 3: Functions & Numerical Methods

(ဖန်ရှင်များနှင့် ခန့်မှန်းတွက်ချက်ခြင်း နည်းပညာ)

3.1 Function (ဖန်ရှင်) ဆိုတာ ဘာလဲ?

သချိုင်္ဘာမှာ f(x) = y လို့ ရေးတာမြင်ရင် မလန့်ပါနဲ့။ Function ဆိုတာ **"စက်" (Machine)** တစ်ခုပါပဲ။

- **Input:** တစ်ဖက်ကနေ ကုန်ကြမ်း (x) ထည့်လိုက်မယ်။
- **Process:** စက်ထဲမှာ တစ်ခုခု လုပ်လိုက်မယ်။
- **Output:** ဟိုဘက်ကနေ ကုန်ချော (y) ထွက်လာမယ် [1]။

ကွန်ပျူတာ ကုဒ်ဥပမာ:

```
def double_machine(x):
    return x * 2
```

ဒီမှာ x=5 ထည့်လိုက်ရင်၊ အဖြေ 10 ထွက်လာမယ်။ ဒါကို သချိုင်္ဘာလို f(x) = 2x လို့ ရေးတာပါ ။

3.2 Numerical Methods (ဂဏန်းဖြင့် ခန့်မှန်းနည်းများ)

တစ်ခါတလေမှာ အဖြေအတိအကျ ရှာဖို့ အရမ်းခက်တဲ့ (သို့မဟုတ်) မဖြစ်နိုင်တဲ့ ညီမျှခြင်းတွေ ရှိပါတယ်။

ဥပမာ $x^5 - 3x + 1 = 0$ လိုမျိုးပေါ့။ ဒါကို ဖြေရှင်းဖို့ ကွန်ပျူတာတွေက "Numerical Methods" (ခန့်မှန်းတွက်ချက်နည်း) ကို သုံးပါတယ်။

အရိုးရှင်းဆုံးနဲ့ အသုံးအများဆုံး နည်းလမ်းတစ်ခုကို ကြည့်ရအောင် -

(a) Bisection Method (ထက်ဝက်ချိုး ရှာနည်း)

ဒါက "ဂဏန်းဘယ်လောက်လဲ မှန်းကြည့်" (Guess the Number) ဂိမ်းနဲ့ တူပါတယ်။

- **ဂိမ်းစည်းကမ်း:** ၁ ကနေ ၁၀၀ ကြား ဂဏန်းတစ်ခု စိတ်ထဲမှတ်ထားပါ။
- **ကွန်ပျူတာ:** "၅၀ လား?"
- **မင်း:** "မဟုတ်ဘူး၊ အဲ့ထက် **ငယ်** တယ်"
- **ကွန်ပျူတာ:** (ဒါဆို ၅၀ နဲ့ ၁၀၀ ကြားကို ဖြတ်ထုတ်လိုက်ပြီ)။ "၂၅ လား?"
- **မင်း:** "မဟုတ်ဘူး၊ အဲ့ထက် **ကြီး** တယ်"

ဒီလိုနည်းနဲ့ အလယ်ကနေ တောက်လျှောက် ထက်ဝက်ချိုးပြီး ရှာသွားရင် အဖြေမှန်ကို အရမ်းမြန်မြန် တွေ့နိုင်ပါတယ်။ ကွန်ပျူတာက ဒီနည်းကိုသုံးပြီး ရှုပ်ထွေးတဲ့ ညီမျှခြင်းတွေရဲ့ အဖြေကို ရှာဖွေပါတယ်။

(b) Newton–Raphson Method (ဆင်ခြေလျှောနည်း)

Bisection ထက် ပိုမြန်တဲ့ နည်းပညာပါ။ ဂရပ် (Graph) ပေါ်က မျဉ်းကွေး (Curve) ရဲ့ ဆင်ခြေလျှော (Slope/Tangent) ကို လိုက်ပြီး အဖြေရှာတာ ဖြစ်ပါတယ်။

- ဒီနည်းက အဖြေနဲ့ နီးစပ်တဲ့ နေရာကနေ စလိုက်ရင်၊ အဖြေမှန်ဆီကို လျှပ်စီးလက်သလို မြန်မြန် ရောက်သွားတတ်ပါတယ်။

လက်တွေ့ ဥပမာ – $\sqrt{10}$ ကို ရှာဖွေခြင်း

ကျွန်တော်တို့က **၁၀** ရဲ့ စတုရန်းရင်း (Square Root of 10) ကို ရှာချင်တယ် ဆိုပါစို့။ (အဖြေမှန်က **3.162277...** ဖြစ်ပါတယ်)။

ဒီနေရာမှာ Newton-Raphson ဖော်မြူလာကို သုံးပါမယ်။

$$x_{new} = x_{old} - \frac{f(x_{old})}{f'(x_{old})}$$

လွယ်ကူအောင် ပြောရရင် -

$$x_{new} = \frac{1}{2}\left(x_{old} + \frac{\text{မူရင်းဂဏန်း}}{x_{old}}\right)$$

အဆင့် (၁) - ပထမဆုံး အကြမ်းဖျင်း ခန့်မှန်းခြင်း (Initial Guess)

၁၀ နဲ့ နီးစပ်ပြီး စတုရန်းတင်လို့လွယ်တဲ့ ဂဏန်းတစ်ခုကို စိတ်မှန်းနဲ့ ရွေးလိုက်ပါ။

- $3^2 = 9$ (၁၀ နဲ့ နီးတယ်)။
- ဒါကြောင့် $x = 3$ ကနေ စလိုက်ကြမယ်။

အဆင့် (၂) - ပထမအကြိမ် တွက်ချက်ခြင်း (Iteration 1)

ဖော်မြူလာထဲ ထည့်တွက်မယ် -

$$x = \frac{1}{2}\left(3 + \frac{10}{3}\right)$$

$$x = \frac{1}{2}(3 + 3.3333)$$

$$x = \mathbf{3.1667}$$

(တွေ့လား? တစ်ခါတည်းနဲ့ အဖြေမှန် 3.162 နဲ့ တော်တော်နီးစပ်သွားပြီ)

အဆင့် (၃) - ဒုတိယအကြိမ် တွက်ချက်ခြင်း (Iteration 2)
ရလာတဲ့ အဖြေ (3.1667) ကို သုံးပြီး ထပ်တွက်မယ် -

$$x = \frac{1}{2}\left(3.1667 + \frac{10}{3.1667}\right)$$

$$x = \frac{1}{2}(3.1667 + 3.1579)$$

$$x = \mathbf{3.1623}$$

နှိုင်းယှဉ်ချက် (Comparison)

- **အဖြေမှန်: 3.162277...**
- **Newton Method (၂ ကြိမ်တွက်): 3.1623** (ဒသမ ၃ နေရာအထိ မှန်သွားပါပြီ!)

ကောက်ချက်

Bisection Method သာဆိုရင် ဒီလောက်မှန်ဖို့ အခါ ၂၀ လောက် အဆင့်ဆင့် ရှာရမှာ ဖြစ်ပေမယ့်၊ Newton-Raphson နည်းကတော့ လျှပ်စီးလက်သလို (Lightning Fast) အဖြေမှန်ကို တန်းရောက်သွားတာ တွေ့ရပါလိမ့်မယ်။

3.3 Modern Applications (လက်တွေ့ကမ္ဘာအသုံးချမှု)

ဒီ "ခန့်မှန်းနည်း" တွေကို ဖုန်းတွေ၊ ဆော့ဖ်ဝဲတွေထဲမှာ နေ့စဉ် သုံးနေကြပါတယ်။

(a) Google Maps (ခရီးကြာချိန် တွက်ခြင်း)

- မင်းအိမ်ကနေ ကျောင်းကိုသွားရင် ဘယ်လောက်ကြာမလဲ?
- လမ်းမှာ မီးပွိုင့်တွေ၊ ကားပိတ်တာတွေ ရှိနိုင်တဲ့ အတွက် အဖြေ "အတိအကျ" (Exact Solution) မရှိနိုင်ပါဘူး။
- Google က Numerical Methods တွေကို သုံး ပြီး "၁၅ မိနစ် ခန့်မှန်းခြေ ကြာမယ်" ဆိုပြီး အဖြေထုတ်ပေးတာ ဖြစ်ပါ တယ်။

(b) Video Games (ရူပဗေဒ တွက်ချက်ခြင်း)

- ဂိမ်းထဲမှာ ဘောလုံးတစ်လုံး ကန်လိုက်ရင် ဘယ်နားကျမလဲ?
- လေတိုက်နှုန်း၊ ကမ္ဘာမြေဆွဲအား တွေကို ထည့်တွက်ဖို့ ညီမျှခြင်း တွေ အများကြီး ဖြေရှင်းရပါတယ်။ ဂိမ်းစက်က တစ်စက္ကန့်အတွင်း ဒီ လို ခန့်မှန်းချက်ပေါင်းများစွာကို တွက်ထုတ်ပေးနေတာ ဖြစ်ပါတယ်။

(c) AI & Machine Learning

- AI ဆိုတာ တကယ်တော့ "အမှား အနည်းဆုံး ဖြစ်အောင် ခန့်မှန်း တဲ့ စက်" ပါပဲ။
- သူက အဖြေမှန်ကို မသိပေမယ့်၊ အဖြေမှန်နဲ့ အနီးစပ်ဆုံး ဖြစ်အောင် (Minimize Error) ညီမျှခြင်းတွေကို အကြိမ်သန်းပေါင်းများစွာ ညှိ ယူ (Optimize) နေတာ ဖြစ်ပါတယ်။

Chapter 4: Linear Programming

(အရင်းအမြစ်များကို စီမံခန့်ခွဲ၍ အမြတ်အများဆုံးရှာခြင်း)

4.1 Linear Programming ဆိုတာ ဘာလဲ?

သင့်မှာ ပိုက်ဆံအများကြီးရှိမယ်၊ အချိန်တွေအများကြီးရှိမယ်၊ ကုန်ကြမ်းတွေ အကန့်အသတ်မရှိဘူးဆိုရင် စီးပွားရေးလုပ်ရတာ အရမ်းလွယ်မှာပါ။ ဒါပေမဲ့ လက်တွေ့ဘဝမှာ **အရာရာဟာ အကန့်အသတ် (Limited Resources)** ရှိပါတယ်။

- ငွေအရင်းအနှီး အကန့်အသတ်ရှိတယ်။
- လူအင်အား အကန့်အသတ်ရှိတယ်။
- အချိန် ၂၄ နာရီပဲ ရှိတယ်။

ဒီလို အကန့်အသတ်တွေကြားထဲကနေ **"အမြတ်အများဆုံး ရအောင် ဘယ်လို လုပ်မလဲ?"** (သို့မဟုတ်) **"ကုန်ကျစရိတ် အသက်သာဆုံးဖြစ်အောင် ဘယ်လိုလုပ်မလဲ?"** ဆိုတာကို တွက်ချက်တဲ့နည်းကို **Linear Programming (LP)** လို့ ခေါ်ပါတယ်။

4.2 လက်တွေ့ဥပမာ (ပရိဘောဂ စက်ရုံ)

မင်းက ပရိဘောဂ စက်ရုံပိုင်ရှင် တစ်ယောက်ဆိုပါစို့။ မင်းစက်ရုံက **စားပွဲ (Table)** နဲ့ **ထိုင်ခုံ (Chair)** နှစ်မျိုးပဲ ထုတ်ပါတယ်။

မင်းမှာရှိတဲ့ အကန့်အသတ်များ (Constraints):

1. **သစ်သား:** ဂိုဒေါင်ထဲမှာ သစ်သား **၁၀၀ ပေ** ပဲ ကျန်တော့တယ်။

2. **လုပ်အား:** အလုပ်သမားတွေက စုစုပေါင်း **နာရီ ၆၀** ပဲ အလုပ်လုပ်နိုင်မယ်။

ထုတ်လုပ်မှု လိုအပ်ချက်:

- **စားပွဲ ၁ လုံး** ထုတ်ရင်: သစ်သား ၅ ပေ၊ လုပ်အား ၂ နာရီ ကုန်မယ်။ အမြတ် **$၂၀** ရမယ်။
- **ထိုင်ခုံ ၁ လုံး** ထုတ်ရင်: သစ်သား ၂ ပေ၊ လုပ်အား ၃ နာရီ ကုန်မယ်။ အမြတ် **$၁၀** ရမယ်။

မေးခွန်း:
အမြတ်အများဆုံးရဖို့ စားပွဲဘယ်နှလုံး၊ ထိုင်ခုံဘယ်နှလုံး ထုတ်မလဲ?
(စားပွဲကြီးပဲ ထုတ်ရင် သစ်သားကုန်မြန်မယ်၊ ထိုင်ခုံကြီးပဲ ထုတ်ရင် လုပ်အား နာရီ ကုန်မြန်မယ်။ မျှထုတ်မှ ရမယ်)

4.3 ကွန်ပျူတာ နားလည်အောင် ဘာသာပြန်ခြင်း

Objective Function: Maximize$P = 10C + 20T$
Constraints:

- $2C + 5T \leq 100$ (Wood)
- $3C + 2T \leq 60$ (Labor)

4.4 အဖြေရှာခြင်း (The Solution)

ကွန်ပျူတာပရိုဂရမ်တွေက ကုမ္ပဏီကြီးတွေအတွက် အမြတ်အများဆုံး ရအောင် ဘယ်လိုတွက်ပေးသလဲ? ဒါကို **Linear Programming** နဲ့တွက်ရပါတယ်။

ဥပမာ - ပရိဘောဂ စက်ရုံ
သင့်မှာ သစ်သား (Wood) နဲ့ လုပ်အား (Labor) အကန့်အသတ်ပဲ ရှိတယ်။

- **ထိုင်ခုံ (Chair)** တစ်လုံးထုတ်ရင် သစ်သား ၂ ပေ၊ လုပ်အား ၃ နာရီ ကုန်မယ်။ အမြတ် ၁၀ ဒေါ်လာရမယ်။
- **စားပွဲ (Table)** တစ်လုံးထုတ်ရင် သစ်သား ၅ ပေ၊ လုပ်အား ၂ နာရီ ကုန်မယ်။ အမြတ် ၂၀ ဒေါ်လာရမယ်။
- သင့်မှာ သစ်သား ပေ ၁၀၀ နဲ့ လုပ်အား နာရီ ၆၀ ပဲ ရှိတယ်။

ကွန်ပျူတာက ဤပြဿနာကို ဖြေရှင်းရန်အတွက် အောက်ပါအဆင့်များအတိုင်း တွက်ချက်ပါသည် -

အဆင့် (၁) – ညီမျှခြင်းများ တည်ဆောက်ခြင်း

- **C** = ထိုင်ခုံ အရေအတွက်
- **T** = စားပွဲ အရေအတွက်
- **အမြတ်** $P\$ = 10C + 20T$ (ဒါကို အများဆုံး ဖြစ်အောင် လုပ်ရပါမည်)

အဆင့် (၂) - ဖြစ်နိုင်ခြေရှိသော အဖြေများကို ရှာဖွေခြင်း

ကန့်သတ်ချက်မျဉ်းကြောင်းများ ဖြတ်သွားသော နေရာများ (Corner Points) ကို ရှာဖွေရပါမည်။

1. **စားပွဲချည်းပဲ ထုတ်မည်ဆိုလျှင် ($C=0$):**
    - သစ်သားအရ: $5T \leq 100 \rightarrow T \leq 20$လုံး
    - လုပ်အားအရ: $2T \leq 60 \rightarrow T \leq 30$လုံး
    - ကောက်ချက်: အများဆုံး **၂၀** လုံးပဲ ထုတ်လို့ရမယ် (သစ်သားကုန်သွားလို့)။
    - အမြတ်: $20 \text{ tables} \times \$20 = \mathbf{400}\$$
2. **ထိုင်ခုံချည်းပဲ ထုတ်မည်ဆိုလျှင် ($T=0$):**
    - သစ်သားအရ: $2C \leq 100 \rightarrow C \leq 50$လုံး
    - လုပ်အားအရ: $3C \leq 60 \rightarrow C \leq 20$လုံး

- ကောက်ချက်: အများဆုံး ၂၀ လုံးပဲ ထုတ်လို့ရမယ် (လုပ်အားကုန်သွားလို့)။
- အမြတ်:$20 \text{ chairs} \times \$10 = \mathbf{200}\$$

3. နှစ်မျိုးမျှပြီး ထုတ်မည်ဆိုလျှင် (Intersection Point): ညီမျှခြင်းနှစ်ခု ဆုံမှတ်ကို တွက်ချက်ရပါမည်။
   - $2C + 5T = 100$ (Eq 1)
   - $3C + 2T = 60$ (Eq 2)
4. ဒီညီမျှခြင်းနှစ်ခုကို ဖြေရှင်းလိုက်တဲ့အခါ $C \approx 9$ လုံး နှင့် $T \approx 16$ လုံး ရရှိပါမည်။
   - စစ်ဆေးခြင်း:
     - သစ်သား: $(2 \times 9) + (5 \times 16) = 18 + 80 = 98$ ပေ (၁၀၀ အောက်မို့ အဆင်ပြေ)
     - လုပ်အား: $(3 \times 9) + (2 \times 16) = 27 + 32 = 59$ နာရီ (၆၀ အောက်မို့ အဆင်ပြေ)
   - အမြတ်: $(9 \times \$10) + (16 \times \$20) = 90 + 320 = \$\mathbf{410}$

အဆင့် (၃) - အဖြေထုတ်ခြင်း (Optimization Result)

တွက်ချက်မှုများအရ -

- စားပွဲသီးသန့် ထုတ်ရင် အမြတ် = $400
- ထိုင်ခုံသီးသန့် ထုတ်ရင် အမြတ် = $200
- **ထိုင်ခုံ ၉ လုံး နဲ့ စားပွဲ ၁၆ လုံး တွဲထုတ်ရင် အမြတ် = $410**

ဒီညီမျှခြင်းတွေကို ဖြေရှင်းပြီး ကွန်ပျူတာက "ထိုင်ခုံ ဘယ်နှလုံး၊ စားပွဲ ဘယ်နှလုံး ထုတ်ရင် အမြတ်အများဆုံး ရမယ်" ဆိုတာကို တွက်ချက်ပေးပါတယ်။ ဒါကို **Optimization** လို့ ခေါ်ပါတယ်။

4.5 ခေတ်သစ် အသုံးချမှုများ

Linear Programming ကို ကုမ္ပဏီကြီးတွေက နေရာတိုင်းမှာ သုံးနေကြပါတယ်။

1. **Shipping (ပို့ဆောင်ရေး):**
- FedEx က ကုန်ကား ၁၀၀ ရှိတယ်။ ပစ္စည်း ၁ သောင်း ပို့ရမယ်။ ဆီ အနည်းဆုံးကုန်ပြီး အမြန်ဆုံးရောက်အောင် ဘယ်ကားကို ဘယ်လမ်း က မောင်းခိုင်းမလဲ?
2. **Airline Crew Scheduling (လေကြောင်းလိုင်း):**
- လေယာဉ်မှူးတွေ၊ လေယာဉ်မယ်တွေကို ဘယ်လေယာဉ်နဲ့ ဘယ်လိုတွဲ ပေးရင် လစာစရိတ် အသက်သာဆုံး ဖြစ်မလဲ?
3. **Diet Planning (အာဟာရ တွက်ချက်ခြင်း):**
- ကြက်ဥ၊ နွားနို့၊ အသား.. ဘာတွေ ဘယ်လောက် စားရင် ဗီတာမင်စုံ ပြီး ဈေးအသက်သာဆုံး ဖြစ်မလဲ?

Chapter 5: Statistics & Probability

(အချက်အလက်များကို သုံးသပ်ခြင်းနှင့် ဖြစ်တန်စွမ်း တွက်ချက်ခြင်း)

5.1 Statistics (စာရင်းအင်း) ဆိုတာ ဘာလဲ?

Statistics ဆိုတာ အချက်အလက် (Data) အများကြီးကို ကောက်ယူပြီး၊ နားလည်လွယ်အောင် အနှစ်ချုပ်လိုက်တဲ့ ပညာရပ်ဖြစ်ပါတယ်။

Data တွေက ဘယ်နားမှာ စုနေသလဲ (Central Tendency) ဆိုတာကို တိုင်းတာဖို့ အဓိက နည်းလမ်း (၃) ခု ရှိပါတယ် -

(a) Mean (ပျမ်းမျှတန်ဖိုး)

- **တွက်နည်း:** ဂဏန်းအားလုံးပေါင်းပြီး အရေအတွက်နဲ့ စားလိုက်တာပါ ။
- **လက်တွေ့ ဥပမာ:** မင်းက ဂိမ်း ၅ ပွဲဆော့တယ်။ အမှတ်တွေက 10, 20, 10, 30, 80 ရတယ်။
- Mean = (10+20+10+30+80) / 5 = 30။
- **အားနည်းချက်:** 80 ဆိုတဲ့ အမှတ်အများကြီး တစ်ပွဲပါလာလို့ ပျမ်းမျှက တက်သွားတာပါ။ တကယ်တမ်း မင်းလက်စွမ်းက 30 မဟုတ်ဘဲ 10-20 ဝန်းကျင်ပဲ ရှိနိုင်ပါတယ်။

(b) Median (အလယ်မှတ်)

- **တွက်နည်း:** ဂဏန်းတွေကို အငယ်ဆုံးကနေ အကြီးဆုံး စီလိုက်ပါ။ အလယ်တည့်တည့်က ဂဏန်းကို ယူပါတယ် ။
- **စီလိုက်မယ်:** 10, 10, 20, 30, 80
- **Median:** 20 (ဒါက မင်းရဲ့ ပုံမှန်လက်စွမ်းအစစ်ကို ပိုပြီး ထင်ဟပ်စေပါတယ်)။

(c) Mode (အများဆုံးပါဝင်မှု)

- **တွက်နည်း:** အကြိမ်ရေ အများဆုံး ပါဝင်နေတဲ့ ဂဏန်းဖြစ်ပါတယ် ။

- **Mode:** 10 (နှစ်ကြိမ်ပါလို့)။
- **စီးပွားရေး ဥပမာ:** ဖိနပ်ဆိုင်မှာ Size 40 က Mode ဖြစ်တယ်ဆိုရင်၊ ဆိုင်ရှင်က Size 40 ကို အများဆုံး မှာယူရပါမယ်။

5.2 Standard Deviation (သွေဖည်မှုနှုန်း)

Data တွေက ပျမ်းမျှ (Mean) ကနေ ဘယ်လောက် ကွာခြားပြန့်ကျဲနေလဲ ဆိုတာကို တိုင်းတာတာကို **Standard Deviation (SD)** လို့ခေါ်ပါတယ် [5]။

- **SD နည်းရင်:** Data တွေက ပျမ်းမျှနားမှာပဲ စုနေတယ် (ငြိမ်တယ်)။
- **SD များရင်:** Data တွေက ဟိုရောက်ဒီရောက် ဖြစ်နေတယ် (မငြိမ်ဘူး)။

ဥပမာ – ဘောလုံးသမား ၂ ယောက်:

- **Player A:** ပွဲတိုင်း ဂိုးသွင်းနှုန်း 1, 1, 1, 1 (Mean = 1, SD = 0) -> ယုံကြည်ရတယ်။
- **Player B:** ဂိုးသွင်းနှုန်း 0, 0, 4, 0 (Mean = 1, SD = High) -> မှန်းရခက်တယ်။

5.3 Probability (ဖြစ်တန်စွမ်း)

Probability ဆိုတာ အနာဂတ်မှာ တစ်ခုခုဖြစ်လာနိုင်ခြေ ဘယ်လောက်ရှိလဲဆိုတာကို တွက်ချက်ခြင်း ဖြစ်ပါတယ်။ တန်ဖိုးက 0 (လုံးဝမဖြစ်နိုင်) ကနေ 1 (သေချာပေါက်ဖြစ်မယ်) ကြားမှာ ရှိပါတယ် ။

$$P(A) = \frac{\text{Number of favorable outcomes}}{\text{Total number of outcomes}}$$

(a) Simple Probability (ရိုးရှင်းသော ဖြစ်တန်စွမ်း)

- **အန်စာတုံး (Die):** ဂဏန်း 6 ကျဖို့ ဖြစ်တန်စွမ်းက 1/6 ပါ။
- **Coin Toss:** ခေါင်း (Head) ကျဖို့ ဖြစ်တန်စွမ်းက 1/2 (သို့မဟုတ် 50%) ပါ ။

(b) Conditional Probability (အခြေအနေအလိုက် ဖြစ်တန်စွမ်း)

တစ်ခုခုဖြစ်ပြီးမှ နောက်တစ်ခု ထပ်ဖြစ်ဖို့ တွက်ချက်တာပါ။ သင်္ကေတက P(B|A) ဖြစ်ပါတယ် (A ဖြစ်ပြီးမှ B ဖြစ်နိုင်ခြေ) ။

AI ဥပမာ – Spam Filter:

- **မေးခွန်း:** ဒီ Email က Spam ဖြစ်နိုင်လား?
- **Condition:** Email ထဲမှာ "Free Money" ဆိုတဲ့ စကားလုံး ပါနေတယ်။
- **Probability:** "Free Money" ပါရင် Spam ဖြစ်နိုင်ခြေ ၉၀% ရှိတယ်။ ဒါကြောင့် AI က Spam Folder ထဲ ပို့လိုက်တာ ဖြစ်ပါတယ်။

5.4 Data Distributions (အချက်အလက် ပုံသဏ္ဌာန်များ)

Data တွေကို ဂရပ်ဆွဲလိုက်တဲ့အခါ ထွက်လာတဲ့ ပုံစံတွေက အရေးကြီးပါတယ်။

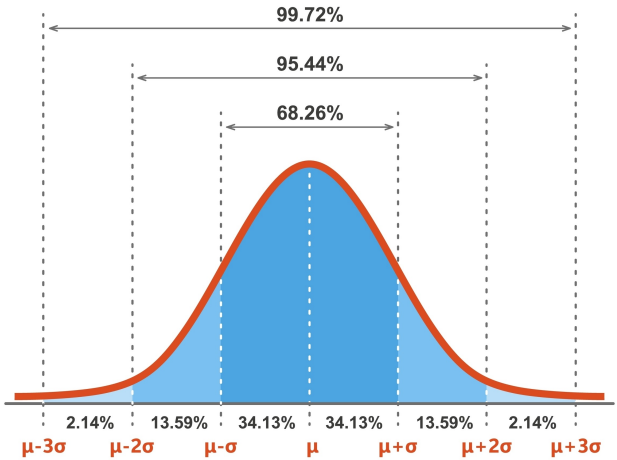
(a) Normal Distribution (ခေါင်းလောင်းပုံ)

သဘာဝတရားမှာ အများဆုံးတွေ့ရတဲ့ ပုံစံပါ။ (ဥပမာ - လူတွေရဲ့ အရပ်အမြင့်၊ စာမေးပွဲ အမှတ်များ)။

- အလယ် (Mean) မှာ လူအများစု ရှိမယ်။
- ထိပ်ဆုံးနဲ့ အောက်ဆုံး (Extremes) မှာ လူနည်းနည်းပဲ ရှိမယ်။

(b) Applications in Data Science

Netflix က မင်းကြိုက်မယ့် ရုပ်ရှင်ကို ဘယ်လို ခန့်မှန်းလဲ?



- သင်နဲ့ အကြိုက်တူတဲ့ လူ ၁၀၀၀ ရဲ့ Data (Distribution) ကို ယူလိုက်တယ်။
- အဲ့ဒီလူတွေက "Action ကား" ကြိုက်ရင် သင်လည်း ကြိုက်နိုင်ခြေ (Probability) များတယ်ဆိုပြီး Recommend ပေးတာ ဖြစ်ပါတယ်။

*"ဤစာအုပ်များသည် တက္ကသိုလ်များတွင် သင်ကြားပို့ချနေသော Computer Science သင်ရိုးညွှန်းတမ်း (Curriculum) များကို အခြေခံထားပါသည်။ သို့သော် ယနေ့ခေတ် နည်းပညာများနှင့် ကိုက်ညီစေရန်အတွက် Python ဘာသာစကား၊ ခေတ်မီ Hardware နည်းပညာများနှင့် လက်တွေ့လုပ်ငန်းခွင်သုံး ဥပမာများကို ဖြည့်စွက် ပြုပြင်ရေးသားထားခြင်း ဖြစ်ပါသည်။

မူရင်းသင်ရိုးကို ပြုစုခဲ့ကြသော ဆရာ/ဆရာမ များအားလုံးကို လေးစားစွာဖြင့် Credit ပေးပါသည်။"*

Yan Naing Lin
Software Engineer