ภาคผนวก

ภาคผนวก ก การใช้รหัสเทียมแสดงขั้นตอนวิธีการทำงาน

ภาคผนวก ข ทบทวนเรื่องเมตริกซ์

ภาคผนวก ค ตัวอย่างโปรแกรมภาษาฟอร์แทรนสำหรับการวิเคราะห์เชิงตัวเลข

ภาคผนวก ก
การใช้รหัสเทียมแสดงขั้นตอนวิธีการทำงาน

Pseudoprograms สำหรับ numerical algorithms จะอยู่ในรูปแบบดังนี้

Algorithm: Name (Comments about Name)
Purpose : (General description, introducing important variables)

pseudocode ซึ่งเขียนตามรูปแบบและวิธีการที่จะกล่าวต่อไป

ตัวอย่าง

Algorithm: Synthetic Division (Horner's Method)
Purpose : To evaluate

$$PolValue = a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}$$

```
GET n, a1 , a, ,..., an+1 , X
PolValue <--- 0
DO FOR i = 1 TO ntl
   PolValue <--- Polvalue * x t a,
```

ตัวอย่างโปรแกรมย่อยซับรูทีน และโปรแกรมย่อยฟังก์ชัน ในภาษาฟอร์แทรน ซึ่งเขียนจากขั้นตอนวิธี (algorithm) ข้างต้น

```
      SUBROUTINE POLY(A,X,M,POLV)
C******M = N+1 (N = DEGREE OF POLYNOMIAL)
      DIMENSION A(M)
      POLV = 0
      DO 10 1. = 1,M
10    POLV = POLV * X t A(I)
      RETURN
      END
```

และ

```
      FUNCTION POLV(A,X,M)
      DIWENSION A(M)
      POLV = 0
      DO 10 I=1,M
10    POLV = POLV * X t A(I)
      RETURN
      END
```

สัญลักษณ์ที่ใช้ใน Algorithm

IDENTIFIERS (Name of Variables)

A. **ตัวแปร** เป็นชื่อที่ใช้เก็บค่าที่เป็นเลขจำนวน จะเขียนด้วยตัวเอน (*italics*) โดยขึ้นต้นด้วย uppercase letter ทั้งนี้ไม่จำกัดจำนวนตัวอักขระที่ประกอบเป็นชื่อ อย่างไรก็ดีตัวแปรชนิด integer ที่ใช้เป็นดรรชนีย่อย (subscript) จะใช้ lowercase letter เช่น i, j, k, m, n

ตัวอย่าง *Value, NumSig, DxMax, Tolerance*

หมายเหตุ ตัวอักษร *S* ใน *NumSig* และ *M* ใน *DxMax* นั้นใช้เพื่อความสะดวกในการอ่าน แสดงให้ทราบว่าเป็นชื่อย่อมาจากคำมากกว่า 1 คำ

B. <u>เวคเตอร์</u> (One-Dimensional Arrays) จะใช้ lowercase letter ตัวหนาแสดงเวคเตอร์เช่น b, c, x_o โดยที่สมาชิกของเวคเตอร์เหล่านี้จะใช้ b_i, c_i, $x_{o\,i}$ (หรือ $x_{o,i}$) ตามลำดับ

b, c, x_o แทน Column Vectors

b', c', x_o' แทน Row Vectors

C. <u>เมตริกซ์</u> (Two-Dimensional Arrays) จะใช้ uppercase letter ตัวหนา เช่น A, B, DD เป็นต้น เพื่อไม่ให้ปะปนกับตัวแปรอื่น ๆ สมาชิกของเมตริกซ์ใช้ a_{ij}, b_{ij}, และ DD(i,j) ตามลำดับ

D. <u>ฟังก์ชัน</u> ใช้ในรูปปกติที่ใช้กันทั่ว ๆ ไปคือ f(x), g(x), f(t,y) เป็นต้น

INPUT

ในการปฏิบัติตามขั้นตอนวิธีหนึ่ง ๆ เมื่อต้องการใช้ค่าต่าง ๆ จะแสดงด้วยคำสั่ง

GET (data list)

<u>ตัวอย่าง</u> เช่นขั้นตอนวิธีต้องการขนาดของเมตริกซ์ A คือ m และ n และสมาชิกของ A และค่า λ(lambda) เราจะเขียนว่า

GET m, n, *A*, *Lambda*

OUTPUT

เราอาจสนใจจะดูค่า(ให้พิมพ์ข้อมูลออก) ของค่าที่คำนวณได้ระหว่างการปฏิบัติตามคำสั่ง และ ค่าคำนวณในขั้นสุดท้าย เราจะใช้คำสั่งแสดงดังนี้

OUTPUT (รายชื่อตัวแปรซึ่งเราต้องการทราบค่า)

OUTPUT (ข้อความเกี่ยวกับค่าของตัวแปรที่เราสนใจ)

<u>ตัวอย่าง</u> เช่น ถ้าเราได้ u จาก x แล้วต้องการให้แสดงทั้ง x และ u เราอาจเขียนดังนี้

OUTPUT (อะเรย์เริ่มต้น: x ; อะเรย์ใหม่: u)

ใน pseudoprogram จะไม่ใส่คำสั่งพิมพ์หัวตาราง หรือส่วนอื่นที่ใช้ปรับปรุงความเข้าใจผลลัพธ์ที่จะต้องแสดง

คำสั่ง OUTPUT จะใส่ไว้เพียงเพื่อแสดงว่าส่วนใดของโปรแกรมที่จะมีผลลัพธ์นั้นเกิดขึ้นแล้ว

ASSIGNMENT STATEMENTS

ลูกศรย้อนกลับ "<---" (back-arrow) จะใช้ในการกำหนดค่าให้แก่ตัวแปร

ตัวอย่าง *Det* <--- *Det* * a_{jj}

มีความหมายว่าให้คูณค่าของ *Det* ด้วยค่าของ a_{jj} แล้วเก็บผลลัพธ์ที่ได้จากการคูณใน *Det* ดังนั้น *Det* จะมีค่าใหม่เท่ากับค่าเดิมคูณด้วย a_{jj}

STATEMENT FORMAT

A. Simple Statements

โดยทั่ว ๆ ไป แต่ละบรรทัดของ pseudoprogram จะใช้เขียนคำสั่งหนึ่งคำสั่ง (statement) คำสั่งจะถูกเขียนเริ่มจากซ้ายสุด ยกเว้นถ้ามีการย่อหน้าเข้ามา (indented) ตามที่จะได้อธิบายต่อไป ถ้าใน 1 บรรทัดมีเกิน 1 คำสั่ง เราจะแยกคำสั่งจากกันด้วยเครื่องหมาย ";"

B. Compound Statements (BEGIN...END)

เมื่อมีคำสั่งหลาย ๆ คำสั่งซึ่งจะถูกปฏิบัติตามกันไปเป็นบล็อก หรือไม่ถูกปฏิบัติตามเลยทั้งบล็อก เราจะล้อมรอบคำสั่งเหล่านั้นไว้ในคำสั่ง BEGIN และ END (ซึ่งมักจะเขียนย่อหน้าเข้ามาทั้ง 2 คำ) หรืออาจเขียนไว้ในบรรทัดเดียวกัน ดังตัวอย่างต่อไปนี้

```
BEGIN a <--- x ; L <--- Y END
```

หรืออาจเขียนในหลาย ๆ บรรทัดดังนี้

```
BEGIN
a <--- x
L <--- Y
END
```

เป็นสิ่งสำคัญที่ต้องคิดว่า compound statement หนึ่ง ๆ นั้นคือคำสั่งหนึ่ง (a single statement) นั่นเอง

FLOW OF CONTROL
(IF statements)

หลาย ๆ ครั้งที่การกระทำหนึ่ง ๆ จะถูกกระทำต่อไปหรือไม่โดยขึ้นอยู่กับว่าเงื่อนไขหนึ่งเป็นจริงหรือไม่ นั่นคือนิพจน์ตรรกะ (logical expression) ซึ่งเป็นเงื่อนไขเป็นจริงหรือเท็จ

A. IF...THEN

โครงสร้างที่ถูกใช้เพื่อที่จะทำให้แน่ใจว่าการกระทำหนึ่งจะถูกกระทำถ้าเงื่อนไขเป็นจริงคือ

IF (logical expression) THEN (statement)

ในที่นี้ (statement) อาจเป็น compound statement ในกรณีนี้เราจะใช้รูปแบบของการย่อหน้าเข้ามา

```
IF (logical expression) THEN
   (statement)
```

ตัวอย่าง เช่น conditional statement

IF *Val* > *MaxVal* THEN *Val* <--- *MaxVal*

คำสั่งนี้ทำให้เราแน่ใจว่า ค่าของ *Val* จะไม่เกินค่าของ *MaxVal* และถ้าเราต้องการใช้ตัวแปรชนิด integer n เพื่อนับจำนวนครั้งที่คำสั่ง *Val* <--- *MaxVal* ถูกกระทำ เราจะเขียนคำสั่งดังนี้

```
IF Val > MaxVal THEN
   BEGIN Val <--- MaxVal; n <--- n+1 END
```

ทั้งนี้เราต้องกำหนดค่าเริ่มต้นให้ n เป็น 0 ก่อน ในตอนเริ่มต้นของโปรแกรมนั่นคือ

```
:
n <--- 0
:
IF Val > MaxVal THEN
   BEGIN
   Val <--- MaxVal
   n <--- n+1
   END
```

B. **IF...THEN...ELSE**

โครงสร้างนี้ถูกใช้เพื่อที่จะทำให้แน่ใจว่าการกระทำหนึ่งจะถูกกระทำเมื่อเงื่อนไขจริง และการกระทำอีกการกระทำหนึ่งจะถูกทำเมื่อเงื่อนไขไม่จริง

รูปแบบของคำสั่งคือ

IF (logical expression)
THEN [statement to be executed if (logical expression, is true]
ELSE [different statement to be executed if (logical expression) is false]

ตัวอย่าง

```
IF L * Y > 0
        THEN Y <--- -Y
        ELSIE BEGIN L <--- 2 * L ; Y <--- 0.5 * Y END
```

นั่นคือถ้า L และ Y มีเครื่องหมายเหมือนกัน เราจะเปลี่ยนเครื่องหมายของ Y และถ้าไม่เป็นดังกล่าวจะเพิ่มค่าของ L เป็น 2 เท่าของค่าเดิม และลดค่าของ Y ลงครึ่งหนึ่ง

REPETITION STATEMENT (Iteration)

A. Count Controlled Loops

ถ้าคำสั่งหนึ่ง (อาจเป็น compound) ถูกกระทำซ้ำ ๆ เท่ากับจำนวนครั้งที่ระบุไว้ เราจะใช้โครงสร้าง

DO FOR ***Index*** = ***Istart*** TO ***Istop***

(statement)

โครงสร้างนี้จะทำให้ (statement) ถูกกระทำตั้งแต่ *Index = Istart*, *Istart* + 1,..., *Istop* (statement) จะไม่ถูกกระทำถ้า *Istart* > *Istop* [ในภาษาฟอร์แทรน 4 (statement) จะถูกกระทำ 1 ครั้ง แม้ว่าเริ่มต้น *Istart* > *Istop* ดังนั้นเพื่อหลีกเลี่ยงต้องเพิ่มคำสั่ง IF(ISTART .GT. ISTOP)GO TO 100 ไว้ก่อน จะเริ่มทำใน loop1

อีกโครงสร้างหนึ่งคือ

DO FOR ***Index*** = ***Istart TO Istop*** STEP ***k***

(statement)

โดยที่ *k* เป็นลบได้ โครงสร้างนี้จะปฏิบัติตาม (statement) เมื่อ *Index = Istart*, *Istart + k,...* จนกระทั่ง *Index* มีค่าเกิน *Istop*

ตัวอย่าง

```
Sum1 <--- 0; Sign <--- +1
DO FOR i = 2 TO 5
   BEGIN
   Sum1 <--- Sum1 t Sign * a,
   Sign <--- - Sign
   END
```

ส่วนของ pseudoprogram นี้จะหา *Sum1 = a, - a, + a, - a,*

ตัวอย่าง

```
Sum2 <--- 0; Sign <--- +1
DO FOR i = 7 TO 2 STEP -2
   BEGIN
   Sum2 <--- Sum2 t Sign * a,
   Sign <--- - Sign
   END
```

ส่วนของ pseudoprogram นี้จะหา $Sum2 = a_7 - a_6 + a_5 - \ldots$

B. Conditionally Controlled Loops

เมื่อเราไม่ทราบจำนวนครั้งของการทำซ้ำล่วงหน้าก่อน เราต้องใช้เงื่อนไข (logical expression) เพื่อควบคุมการทำซ้ำ มีวิธี 2 วิธีที่จะใช้คือ

วิธีที่ 1

```
DO WHILE (logical expression)
   (statement)
```

คำสั่งดังกล่าวจะทำให้คำสั่งต่อไปนี้ถูกทำซ้ำ

```
IF (logical expression) is true
   THEN execute (statement)
   ELSE exit loop
```

วิธีที่ 2

```
DO UNTIL (logical expression)
   (statement)
```

คำสั่งดังกล่าวจะทำให้คำสั่งต่อไปนี้ถูกทำซ้ำ

```
Execute (statement)
IF (logical expression) is true
   THEN exit loop
```

โครงสร้าง DO WHILE นั้นคำสั่งจะไม่ถูกกระทำเลย (no iterations) ถ้า (logical expression) เป็นเท็จเมื่อเริ่มทำคำสั่ง DO แต่สำหรับโครงสร้าง DO UNTIL นั้น คำสั่งจะถูกกระทำอย่างน้อย 1 ครั้ง เพื่อแสดงความแตกต่างนี้ สมมุติว่าเราต้องการหาค่า *GreatInt* เป็นเลขจำนวนเต็มตัวที่มีค่าสูงสุดซึ่งจะมีค่าน้อยกว่าหรือเท่ากับ (<=) เลขจำนวนจริงบวกที่กำหนดให้คือ *x*

ส่วนของโปรแกรมทั้งสองคือ

```
GreatInt <--- 0
DO WHILE GreatInt <= x-1
   GreatInt <--- GreatInt + 1
```

และ

```
GreatInt <--- 0
DO UNTIL GreatInt > x-1
GreatInt <--- GreatInt + 1
```

ทั้งสองวิธีจะคำนวณ *GreatInt* ถูกต้องสำหรับ $x >= 1$ อย่างไรก็ดีถ้า $0 < x < 1$ แล้ว DO WHILE จะให้ค่า *GreatInt* = 0 (ซึ่งถูกต้อง) ในขณะที่ DO UNTIL ให้ค่า *GreatInt* = 1 (ซึ่งไม่ถูกต้อง)

C. Count and Conditionally Controlled Loops

วิธีวิเคราะห์เชิงตัวเลขส่วนมากต้องการกระทำซ้ำจนกว่าจะได้ความแม่นยำตามต้องการ อย่างไรก็ดีเพื่อป้องกันการเกิดลูปส์อนันต์ที่เราไม่ได้คาดไว้ก่อน จึงจำเป็นต้องกำหนดขีดจำกัดสูงสุดเรียก *MaxIt* ให้แก่จำนวนครั้งของการทำซ้ำ ซึ่งเราจะใช้โครงสร้างต่อไปนี้

```
DO FOR k = 1 TO MaxIt UNTIL termination test is satisfied
   BEGIN
   (statement describing the repeated calculation)
   {termination test: (logical expression))
   END
```

การออกจากลูป (loop) จะเกิดขึ้นหลังการทำซ้ำ *MaxIt* ครั้ง หรือก่อนหน้านั้นถ้า termination test ได้ผลเป็นที่พอใจ นั่นคือถ้า (logical expression) เป็นจริง ระหว่างการปฏิบัติการในลูป

DESCRIPTIVE (Informal) STATEMENTS and PHRASES

วลีในภาษาอังกฤษ (นอกเหนือจากโครงสร้างที่กล่าวมาข้างต้น) ซึ่งเขียนตามแบบปกติเช่น

GET termination test parameters

จะถูกใช้ถ้าธรรมชาติที่แน่นอนของพารามิเตอร์ของการทดสอบเพื่อหยุดการทำซ้ำนั้นจะปล่อยไว้ให้ผู้ใช้อธิบายรายละเอียดเอาเองทีหลัง ทำนองคล้ายคลึงกัน คำสั่ง

หาผลเฉลยของ Ax = b คือหา $\bar{x} = A^{-1}b$

ซึ่งละวิธีการหาคำตอบให้ผู้ใช้เอง

COMMENTS AND LABELS

คำอธิบายเพิ่มเติมจะใส่ไว้ในวงเล็บปีกกา ({...})

ตัวอย่าง เช่น

IF L * Y > 0 {L และ Y มีเครื่องหมายเหมือนกัน}

ซึ่งทำให้หมดข้อสงสัยเกี่ยวกับวัตถุประสงค์ของนิพจน์ตรรกะ L * Y > 0 สิ่งที่จะใส่ในวงเล็บปีกกาอาจเป็น labels ซึ่งเป็นวลีย่อ ๆ เขียนด้วย*ตัวเอนหนา* หรือ ตัวหนา เราใช้เพื่อแสดงโครงร่าง (outline) ของขั้นตอนหลัก ๆ ของขั้นตอนวิธี

ตัวอย่าง วิธีวิเคราะห์เชิงตัวเลขส่วนใหญ่จะใช้ขั้นตอนวิธีสำหรับการทำซ้ำในรูป

```
{initialize}
(Statements preparing for the iteration)
{iterate}
DO FOR k = 1 TO MaxIt UNTIL termination test is satisfied
   BEGIN
   (Iterative step of the method, possibly using
    other labels)
   {termination test: (logical expression))
   END
OUTPUT (results of the iteration)
```

นอกไปจาก **termination test** ที่จะถูกกระทำทุกครั้งก่อนจบการทำซ้ำแล้ว comments และ **labels** ในวงเล็บปีกกานั้นไม่ถือเป็นส่วนหนึ่งของ pseudoprogram เราใส่ไว้เพื่อช่วยการอ่านโปรแกรมและเราอาจเอาออกได้โดยที่ไม่มีผลต่อการปฏิบัติงานของขั้นตอนวิธี

STOP STATEMENT
(Algorithm failure)

เราจะใช้คำสั่ง STOP ใน pseudoprogram ถ้าพบว่าระหว่างการปฏิบัติงานในขั้นตอนวิธีอยู่นั้น ขั้นตอนวิธีไม่ให้ผลเฉลยที่ต้องการ ให้ผู้อ่านได้ใช้คำสั่งนี้เพื่อบันทึกเรื่องที่เกิดขึ้น (set a flag and/or print an error message) แล้วหยุดการปฏิบัติงานอย่างเหมาะสมเอาเอง