บทที่ 1
คอมพิวเตอร์ ความคลาดเคลื่อน และขั้นตอนวิธี

1.1 **วิธีวิเคราะห์เชิงตัวเลข (Numerical Method) คืออะไร ?**

1.2 **เครื่องคำนวณแบบดิจิตัล (digital devices) ทำผิดหรือไม่ ?**

1.3 **เครื่องคอมพิวเตอร์เก็บเลขจำนวนจริง (real numbers) อย่างไร ?**

1.4 **ความคลาดเคลื่อนในการคำนวณแบบ Fixed-Precision**

1.5 **การหลีกเลี่ยงความคลาดเคลื่อนจากการปัดเศษ (Roundoff Error)**

1.6 **ความคลาดเคลื่อน ความแม่นยำ และ การทดสอบเพื่อตรวจความใกล้เคียง**

1.7 **ขั้นตอนวิธีของการทำซ้ำ (Iterative Algorithms)**

บทที่ 1
คอมพิวเตอร์ ความคลาดเคลื่อน และขั้นตอนวิธี

1.1 <u>วิธีวิเคราะห์เชิงตัวเลข (Numerical Method) คืออะไร ?</u>

ในวิชาแคลคูลัสเบื้องต้นเราเรียนวิธีการหาอนุพันธ์และการอินทิเกรทเพื่อให้ได้**ผลเฉลยแม่นตรง** (exact answers) แต่โชคไม่ดีที่เทคนิคเบื้องต้นทางแคลคูลัส (และไม่ว่าจะขั้นสูง) เพียงอย่างเดียวนั้นไม่พอเพียงสำหรับการแก้ปัญหาแคลคูลัสตามตัวอย่างต่อไปนี้

<u>ปัญหาที่ 1</u> หา maximum และ minimum values ซึ่งอยู่ในช่วง $[0,1]$ ของ

$$F(x) = x^6 + 5x^4 - 9x + 1$$

ในการทำนั้นเราต้องแก้สมการ polynomial degree 5 นั่นคือ

$$dF(x)/dx = F'(x) = 6x^5 + 20x^3 - 9$$

<u>ปัญหาที่ 2</u> หาค่า $\int_0^b e^{-x^2}\,dx$ และ $\int_0^b \sin(x)/x\,dx$ สำหรับค่า $b > 0$

<u>ปัญหาที่ 3</u> หา $y = y(t)$ ซึ่งสอดคล้องกับสมการเชิงอนุพันธ์

$dy/dx = y^2 + t^2$, $y(0) = 1$ (i.e., $y = 1$ at $t = 0$)

<u>ปัญหาที่ 4</u> หา local extrema ของฟังก์ชันของตัวแปร 2 ตัว

$$f(x,y) = xe^y - \ln(2x^2 + y^2) \quad [\ln = \log_e]$$

ซึ่งต้องทำโดยการหารากของสมการ 2 สมการต่อไปนี้คือ

$$\partial f/\partial x = e^y - [4x/(2x^2+y^2)] = 0$$

$$\partial f/\partial y = xe^y - [2y/(2x^2+y^2)] = 0$$

ถ้าเราลองแก้ปัญหาที่ 1-4 โดยวิธีที่เรียนมา จะพบว่าเป็นไปไม่ได้ที่จะหา

สูตรที่ชัดเจนซึ่งจะใช้กับแต่ละปัญหาเพื่อให้ได้ค่าตอบที่ถูกต้อง

ในสถานการณ์จริงๆนั้นเรามักไม่ต้องการ**ผลเฉลยแม่นตรง** แต่ต้องการผลเฉลยซึ่งมีความแม่นยำถึงระดับที่ต้องการเท่านั้น เราจึงใช้วิธีการวิเคราะห์เชิงตัวเลขช่วยในการหาค่าตอบของปัญหาเช่นดังกล่าวข้างต้นได้

1.1A <u>วิธีวิเคราะห์เชิงตัวเลข และการทดสอบเพื่อหยุดการทำงาน (Termination Tests)</u>

คำ **Algorithm** ถูกใช้เพื่ออธิบายวิธีการเป็นขั้นๆ (step-by-step) ซึ่งต้องใช้จำนวนขั้นซึ่งมีจำนวนจำกัด (finite number of step)

วิธีการวิเคราะห์เชิงตัวเลข (Numerical method) **คือ ขั้นตอนวิธี** (Algorithm) หนึ่งสำหรับการหาค่าเชิงตัวเลขซึ่งมีระดับความแม่นยำตามระดับที่ต้องการ

<u>ตัวอย่าง</u> a. จงอธิบายวิธีวิเคราะห์เชิงตัวเลขสำหรับการหาอนุพันธ์ของฟังก์ชัน f ณ จุด x โดยให้มีความแม่นยำถึงทศนิยมตำแหน่งที่ 4* (four-decimal-place accuracy)

b. จงใช้วิธีวิเคราะห์เชิงตัวเลขสำหรับการประมาณ $d \sin x/dx \big|_{x=0}$ โดยให้มีความแม่นยำถึงทศนิยมตำแหน่งที่ 4

---

* เมื่อกล่าวว่า " X ประมาณค่า x ด้วยความแม่นยำถึงทศนิยมตำแหน่งที่ d " (X approximates x to d decimal places) หมายความว่า X และ x ต่างกันน้อยกว่า 1/2 unit ในทศนิยมตำแหน่งที่ d

นั่นคือ $|X-x| < (1/2)10^{-d}$

ตัวอย่างเช่น x = 12.<u>45672</u>, X = 12.<u>45676</u>

$|X-x| = .00004 < .5 * .0001 = .00005$

แสดงว่า X ประมาณค่า x ด้วยความแม่นยำถึงทศนิยมตำแหน่งที่ 4

Solution

a. เมื่อเราไม่มีสูตรทั่วไปสำหรับการหาอนุพันธ์ เราจะพิจารณา

คำจำกัดความของ

$$f'(x) = df(x)/dx = \lim_{h \to 0} \Delta f(x)/h$$

(if it exists)

โดยที่ $\Delta f(x)/h = [f(x+h)-f(x)]/h$ (difference quotient of f at x)

วิธีแก้ปัญหาแบบปกติคือ ประมาณ f'(x) ด้วย $\Delta f(x)/h$ สำหรับ h เล็กๆ ค่าของ h ที่เล็กพอจะทำให้แน่ใจว่า $\Delta f(x)/h$ ประมาณค่า f'(x) ถึงทศนิยมตำแหน่งที่ 4 นั้นขึ้นอยู่กับฟังก์ชัน ที่กำหนดให้ และจุด x ที่ระบุ เพื่อเป็นการช่วยกำหนดค่าที่เหมาะสมของ h เราเพียงแต่หาค่า $\Delta f(x)/h$ สำหรับค่า h ที่ลดลงเป็นลำดับ และหยุดเมื่อ $E(h) = f'(x) - \Delta f(x)/h$ มีค่าเข้าใกล้ 0 [E(h) คือความคลาดเคลื่อนของการประมาณ]

ถึงแม้วิธีการข้างต้นจะดูง่ายๆ แต่เราต้องทราบค่าของ f'(x) ซึ่งจริงๆแล้วเราไม่ทราบค่าเพราะเรากำลังจะหาค่าของ f'(x) และถ้าเราทราบเราก็ไม่ต้องใช้วิธีวิเคราะห์เชิงตัวเลขเข้าช่วย ดังนั้นเราจะทำการหาค่า $\Delta f(x)/h$ ต่อไปเรื่อยๆ จนกว่าค่านี้จะ **"close enough"**

เนื่องจากว่าเราต้องคำนวณ $\Delta f(x)/h$ ซึ่งสำหรับค่าต่างๆของ h's จะได้แสดงวิธีดังกล่าวในโปรแกรม DERIV

ในการใช้ DERIV เราต้องระบุ **initial h, a shrinking ratio r** และ **จำนวนครั้งที่ต้องการทำซ้ำ** ของขั้นต่อไป

Calculate and print h and $\Delta f(x)/h$

h <-- h/r { replace h by 'current h)/r }

```
C*****DERIV: A FORTRAN PROGRAM FOR ESTIMATING F'(X)
C     APPROXIMATE F'(X) BY DQ(H), H=HO,HO/R,...,
C     HO/R**(NREPS-1)
      F(X)=SIN(X)          <-- Statement function
      WRITE(6,1)
1     FORMAT(1X,'ENTER X,HO,SHRINKING RATIO.8 REPETITIONS')
      READ(5,*)X,H,R,NREPS
      WRITE(6,2)
2     FORMAT(1X,'      H',11X,'DQ(H)')
      DO 10 N =1,NREPS
      DQ=(F(X+H)-F(X))/H
      WRITE(6,3)H,DQ
3     FORMAT(E13.5,2X,F11.6)
      H=H/R
10    CONTINUE
      STOP
      END
```

__Solution__

b. เราพบปัญหาในการหาค่า h ซึ่งต้องเล็กพอที่จะทำให้ $\Delta f(x)/h$ ประมาณ $f'(x)$ ได้ตามความแม่นยำที่ต้องการ

เมื่อ $f(x) = \sin x$ และ $x = 0$

ถ้าเราเลือก $H_{initial} = 1$

R : shrinking ratio 4

และ NREPS = 8

run DERIV ด้วยค่าข้างต้นได้ผลดังนี้

ENTER X,HO,SHRINKING RATIO,# REFETTTIONS

0,1,4,8 column นี้หาเองเพราะทราบค่า F'(X)=1

	H	DQ(H)	E(H)=F'(X)-DQ(H)	
	0.10000E+01	0.841471	0.156529	
	0.25000E+00	0.989616	0.010384	
1/16 =	0.62500E-01	0.999349	0.000651	= E(1/16)
1/64 =	0.15625E-01	0.999959	0.000041	= E(1/64)
1/256 =	0.39063E-02	0.999997	0.000003	
	0.97656E-03	1.000000	0.000000	
	0.24414E-03	1.000000	0.000000	
	0.61035E-04	1 .000000	0.000000	

h = 1/16 = 0.0625 -->DQ(h) = 0.999:; (เลาทศนิยม 4 ตำแหน่ง)

h = 1/64 = 0.015625 -->DQ(h) = 1.0000 (เลาทศนิยม 4 ตำแหน่ง)

ดังนั้น h = 1/64 จึงเล็กพอที่จะทำให้แน่ใจว่า Δf(x)/h ประมาณ sin'(0) ได้แม่นยำถึงทศนิยมตำแหน่งที่ 4

(*)โปรดสังเกตว่า

DQ(1/64) -DQ(1/16)=0.999959-0.999349=0.000610 ≈ E(1/16)

DQ(1/256)-DQ(1/64)=0.999997-0.999959=0.000038 ≈ E(1/64)

โดยทั่วๆไป ผลต่างระหว่างค่าประมาณที่ต่อเนื่องกัน

นั่นคือ ΔDQ = DQ(h) - DQ(preceding h)

คือเลขจำนวนซึ่งสามารถหาได้โดยไม่ทราบ f'(x) และจะถูกใช้เพื่อประมาณค่า error of DQ(preceding h) สิ่งที่ตามมาคือ แทนที่จะต้องปฏิบัติซ้ำทั้ง NREPS ครั้ง **เราจะหยุดกระทำเมื่อขนาดของ ΔDQ มีค่าเล็กมาก**

(**) เงื่อนไขที่เหมาะสมสำหรับ four-place accuracy คือ

$$\left|\Delta DQ\right| < 10^{-4}$$

เมื่อพิจารณา (*) แล้ว จะพบว่า **termination test** (**) จะเป็นเหตุให้หยุดการทำซ้ำหลังจากพิมพ์ H = 1/256 และ DQ = 0.999997 แล้ว

1.1B <u>ทำไมต้องใช้ Programmable Device สำหรับการวิเคราะห์เชิงตัวเลข ?</u>

จากตัวอย่างที่ผ่านมาได้แสดงให้เห็นว่าลักษณะ 3 ประการของคอมพิวเตอร์ (หรือ programmable calculator) ดังต่อไปนี้เหมาะสมกับวิธีวิเคราะห์เชิง-ตัวเลข คือ

1. ความยืดหยุ่น (Flexibility)

โปรดสังเกตว่าเราสามารถเปลี่ยน F(X) ในบรรทัดที่ 4 ของโปรแกรม DERIV ได้ เมื่อโปรแกรมนี้ถูกเก็บในหน่วยความจำ เราประมาณ f'(x) ที่ค่าของ x ใดๆด้วยความแม่นยำถึงทศนิยมตำแหน่งที่ต้องการได้ง่ายๆโดยการเปลี่ยน F(X) ตามต้องการ แล้ว enter ค่าต่างๆของ H, R และ NREPS ระหว่างการปฏิบัติงาน

2. ความมีประสิทธิภาพ (Efficiency)

ขั้นตอนวิธีใน DERIV เป็นแบบอย่างของขั้นตอนของวิธีการวิเคราะห์เชิงตัวเลขในแง่ที่ว่ากระบวนการหนึ่ง (calculate the next DQ, output H and DQ แล้ว shrink H) ถูกกระทำจนกระทั่งนิพจน์ที่ใช้เป็นเงื่อนไขของการหยุดทำงานเป็นจริง ขั้นตอนวิธีดังกล่าวถูกเรียกว่า **ขั้นตอนวิธีสำหรับการทำซ้ำ ("iterative" algorithm)** ในการเขียนโปรแกรมอย่างมีประสิทธิภาพนั้นทำได้โดยใช้ **loop** ดังในตัวอย่าง

โปรแกรม DERIV ตั้งแต่

```
        DO 10 N = 1,NREPS

           :

  10    CONTINUE
```

3. ความเร็ว และ ความเชื่อมั่น (Speed and Reliability)

หลังจากที่โปรแกรมถูกแก้ไขแล้วและให้ข้อมูลเข้าที่จำเป็น โปรแกรมจะถูกปฏิบัติตามด้วยความเร็วที่เร็วกว่าและเชื่อถือได้มากกว่าที่คนคนใดคนหนึ่งจะทำได้

1.2 <u>เครื่องคำนวณแบบดิจิตัล (Digital Devices) ทำผิดหรือไม่ ?</u>

Digital devices often give errorneous answers.

บ่อยครั้งหรือเกือบทั้งหมดของ **คำตอบที่ไม่ถูกต้อง** เกิดจาก human error เช่น

เกิดจาก Programmer กำหนดตรรกะ (logic) ผิดในโปรแกรม

เกิดจาก Keypuncher พิมพ์โปรแกรมผิดหรือข้อมูลเข้าผิด

เกิดจาก User ให้ข้อมูลเข้าผิดๆ ใช้สูตรหรือวิธีการที่ผิด

ความคลาดเคลื่อนทั้งหลายนั้นแก้ได้โดยการ**ค้นหาให้พบแล้วแก้ให้ถูกต้อง**

1.2A <u>ตัวอย่างของความคลาดเคลื่อนซึ่งเกิดจากเครื่องคำนวณแบบดิจิตัล</u>

<u>**ตัวอย่าง**</u>

ให้ $A = (1-\cos x)/x^2$, $B = \sin^2 x/[x^2(1+\cos x)]$

ถ้าคูณ A ด้วย $(1+\cos x)/(1+\cos x)$ จะเห็นว่า A และ B เท่ากัน สำหรับ x ที่ไม่เป็น 0 ทั้งนี้ $\cos x \neq -1$

If $x = 0 + 0.1$, then $A = B = 0.499583$; if $x = \pi + 0.1$, then $A = B = 0.189857$
If $x = 0 + 0.01$, then $A = B = 0.499996$; if $x = \pi + 0.01$, then $A = B = 0.201353$
If $x = 0 + 0.001$, then $A = B = 0.500000$; if $x = \pi + 0.001$, then $A = B = 0.202513$
If $x = 0 + 0.0001$, then $A = B = 0.500000$; if $x = \pi + 0.0001$, then $A = B = 0.202629$
If $x = 0 + 0.00001$, then $A = B = 0.500000$; if $x = \pi + 0.00001$, then $A = B = 0.202641$

ค่า A และ B ที่แสดงไว้ ได้ถูกปัดเศษให้เหลือทศนิยมเพียง 6 ตำแหน่ง

ต่อไปนี้เป็นตัวอย่างโปรแกรมที่ทำการหาค่าของ A และ B จากค่าของ x ทั้ง 10 ค่าข้างต้น

```
C * * * * * * * * ERROR * * * * * * * * * * * *
C  THIS PROGRAM DEMONSTRATES DIGITAL DEVICE ERRORS
C
      DATA IW, IR /5, 5/
      WRITE(IW,1)
    1 FORMAT('0ENTER THE LIMITING VALUE OF X')
      READ(IR,*) XLIMIT
C
      H = 0.1
      DO 10 K=1, 5
         X = XLIMIT + H
C
         A = (1 - COS(X))/X**2
         B = (SIN(X)/X)**2/(1 + COS(X))
C
         WRITE (IW,2) X, A, B
    2    FORMAT(' IF X=',F9.6,' THEN A=',E13.6,
     &          ' AND B=',E13.6)
         H = H/10
   10 CONTINUE
      STOP
      END

ENTER THE LIMITING VALUE OF X
0

IF X= 0.100000 THEN A= 0.499584E+00 AND B= 0.499583E+00
IF X= 0.010000 THEN A= 0.500008E+00 AND B= 0.499996E+00
IF X= 0.001000 THEN A= 0.506639E+00 AND B= 0.500000E+00
IF X= 0.000100 THEN A= 0.745058E+00 AND B= 0.500000E+00
IF X= 0.000010 THEN A= 0.000000E+00 AND B= 0.500000E+00

ENTER THE LIMITING VALUE OF X
3.14159265

IF X= 3.241593 THEN A= 0.189857E+00 AND B= 0.189857E+00
IF X= 3.151593 THEN A= 0.201353E+00 AND B= 0.201346E+00
IF X= 3.142593 THEN A= 0.202513E+00 AND B= 0.199852E+00
IF X= 3.141693 THEN A= 0.202629E+00 AND B= 0.135837E+00
%FRSAPR Floating divide check PC= 235

IF X= 3.141603 THEN A= 0.202641E+00 AND B= 0.170141E+39
```

รูป 1.2-1 โปรแกรมภาษาฟอร์แทรนเพื่อหาค่า A และ B ใกล้ 0 และ π

จากตัวอย่างการวิ่งโปรแกรมจะเห็นว่า คอมพิวเตอร์คำนวณค่าของ B ได้แม่นยำเมื่อ x มีค่าใกล้ 0 และคำนวณค่า A ได้แม่นยำเมื่อ x มีค่าใกล้ π และให้ค่าที่ใช้ไม่ได้เมื่อคำนวณค่า A จากค่า x ใกล้ 0 และคำนวณ B จากค่า x ใกล้ π

ดังนั้นจากตัวอย่างแสดงว่า

Computers make no judgment about the correctness of the values they produce; they will display nonsensical answers as neatly and authoritatively as correct ones.

เราจะประสบปัญหา ถ้าเราเชื่อว่า A = 0 เมื่อ x = 0.00001 เพราะว่าคอมพิวเตอร์ให้คำตอบเช่นนั้น

1.2B ความคลาดเคลื่อนในการใช้สูตรกำลังสอง (Quadratic Formula)

ในการหาราก 2 รากของสมการกำลังสอง $ax^2 + bx + c = 0$ $(a \neq 0)$ โปรแกรมและตัวอย่างการวิ่งโปรแกรมต่อไปนี้แสดงความคลาดเคลื่อนที่อาจเกิดขึ้นเมื่อหารากของสมการข้างต้น

```
C * * * * * * * * * * * * *   QUAD   * * * * * * * * * * * * *
C  THIS PROGRAM DEMONSTRATES ERRORS THAT CAN OCCUR WHEN USING
C  THE QUADRATIC FORMULA TO FIND THE REAL ROOTS OF A*X**2+B*X+C
C
      DATA IW, IR /5, 5/
      WRITE(IW,1)
    1 FORMAT('0FOR REAL ROOTS OF AX**2+BX+C, INPUT A, B, C')
      READ(IR,*) A, B, C
C
      B1 = -0.5*B/A
      C1 = C/A
      DISC = B1*B1 - C1
      IF (DISC .GE. 0.) GOTO 20
         WRITE(IW,2)
    2    FORMAT(' ROOTS ARE COMPLEX')
         STOP
C
C     REAL ROOTS: FIND THEM AS R(+), C1/R(+) AND AS R(-), C1/R(-)
   20 D1 = SQRT(DISC)
      RPLUS  = B1 + D1
      RMINU  = B1 - D1
      R2PLUS = C1/RPLUS
      R2MINU = C1/RMINU
C
      WRITE(IW,3) RPLUS, RMINU, R2PLUS, R2MINU
    3 FORMAT(' ROOTS:     R(+) = ',E13.6,5X,'   R(-) = ',E13.6,
     &       /'        C1/R(+) = ',E13.6,5X,'C1/R(-) = ',E13.6)
      STOP
      END
```

รูป 1.2-2 โปรแกรมภาษาฟอร์แทรนเพื่อคำนวณ $r(+)$, $r(-)$, $c_1/r(+)$, และ $c_1/r(-)$

```
FOR  REAL  ROOTS  OF  AX**2+BX+C,  INPUT  A,   B,  C
2,  9,  -5

ROOTS:        R(+)  =  0.500000E+00        R(-)  =  -0.500000E+01
            C1/R(+) =  -0.500000E+01     C1/R(-)  =   0.500000E+00

FOR  REAL  ROOTS  OF  AX**2+BX+C,  INPUT  A,   B,  C
1,  -100.01,~

ROOTS:        R(+)  =  0.100000E+03        R(-)  =  0.100002E-01
            C1/R(+)  =  0.100000E-01     C1/R(-)  =  0.999977E+02

FOR  REAL  ROOTS  OF  AX**2+BX+C,  INPUT  A,   B,  C
1,  +100.01,  1

ROOTS:        R(+)  =  -0.100002E-01       R(-)  =  -0.100000E+03
            C1/R(+)  =  -0.999977E+02     C1/R(-)  =  -0.100000E-01
```

ในที่นี้ b_1 $-b/2a$, $c_1 = c/a$ และ $d_1 = \sqrt{b_1^2 - c_1}$

$r(+) = b_1 + d_1$, $r(-) = b_1 - d_1$

$r(+)r(-) = b_1^2 - d_1^2 = c_1$

ดังนั้น $r(+) = c_1/r(-)$. $r(-) = c_1/r(+)$ ทั้งนี้ตัวหารต้อง $\neq 0$

จากตัวอย่างที่ผ่านมาใน 1.2A และ 1.2B ความคลาดเคลื่อนที่เกิดขึ้นจะไม่เป็นเอกลักษณ์ (unique) สำหรับคอมพิวเตอร์ทุกเครื่อง

เราต้องทราบที่มาของความคลาดเคลื่อนแล้วแก้ไขหรือทำให้เกิดน้อยที่สุด และเราต้องรู้วิธีการเก็บเลขจำนวนจริงในหน่วยความจำของคอมพิวเตอร์ รวมทั้งการคำนวณโดยใช้คอมพิวเตอร์ที่ทำการคำนวณด้วย Fixed-Precision

1.3 เครื่องคอมพิวเตอร์เก็บเลขจำนวนจริง (Real Numbers) อย่างไร ?

1.3A เลขฐานสอง (Binary Numbers)

เลขฐานสองจำนวนหนึ่งคือเลขจำนวนจริง X ซึ่งอาจถูกเขียนได้ในรูป

$$X=\pm[\underbrace{(b_n2^n+b_{n-1}2^{n-1}+\ldots+b_12^1+b_0)}_{\text{integer part ของ } X}]+[\underbrace{(b_{-1}2^{-1}+b_{-2}2^{-2}+\ldots+b_{-k}2^{-k})}_{\text{fractional part ของ } X}]$$

.....(*)

โดยที่ b_i คือ bit (binary digit) ซึ่งอาจเป็น 0 หรือ 1

นิพจน์ (*) เราเรียกว่า **binary expansion of X** หรือเรียกย่อ ๆ ว่า **binary representation**

ดังนั้น

$$X=\pm(\underbrace{b_nb_{n-1}\ldots b_1b_0}_{(n+1)\text{-bit integer}}\overset{\text{binary point}}{.}\underbrace{b_{-1}b_{-2}\ldots b_{-k}}_{k\text{-bit fraction}})_2 \qquad \text{ทุก } b_i \text{ มีค่า 0 หรือ 1}$$

ตัวอย่าง

$$5.25 = 5\ 1/4 = \underbrace{+[(1.2^2+0.2^1+1)+(0.2^{-1}+1.2^{-2})]}_{\text{binary expansion}} = \underbrace{+(101.01)_2}_{\text{binary representation}}$$

$(\underbrace{11\ldots1}_{m \text{ ตัว}})_2 = 2^m-1$ เป็น the largest m-bit integer

และ $(0.\underbrace{11\ldots1}_{k \text{ ตัว}})_2 = 1-2^{-k}$ เป็น the largest k-bit fraction

ตัวอย่าง

$(11111)_2 = 2^5-1 = 31$ เป็น the largest 5-bit integer

และ $(0.111)_2 = 1-2^{-3} = 7/8$ เป็น the largest 3-bit fraction

1.3B การเก็บเลขจำนวนเต็ม (Integers)

1 word = 32 bits

เลขจำนวนเต็ม X ที่จะเก็บได้ในที่ 32 bits คือ

$$|X| <= (2^{31}-1) = 2,147,483,647$$

การพยายามที่จะเก็บเลขจำนวนเต็มที่มีขนาดใหญ่กว่า $(2^{31}-1)$ จะทำให้เกิด **integer overflow**

bit no.	31	30	29	28	27	26	25	24	23	22	21	20	. . .	3	2	1	0
	0	0	0	1	0	1	1	0	0	1	0	0		0	0	0	1

← 31 bits for the integer's binary representation →

sign bit (0 for +, 1 for -)

รูปแสดงการเก็บเลขจำนวนเต็มในที่ขนาด 32 bits

1.3C การเก็บ Floating-Point Representations ของเลขฐานสอง (Binary Numbers)

ทุกๆ nonzero binary number X ถูกแสดงได้ด้วย

$X = \pm M.2^c$ โดยที่ $M = +(0.1b_{-2}b_{-3}\ldots b_{-k})_2$

และ c = เลขจำนวนเต็มตัวหนึ่ง

เรียกว่า normalized floating-point (binary) representation ของ X

ตัวอย่าง

ตาราง 1.3-1 การเขียนในรูป Normalized Floating-Point

X	Binary Representation	Normalized Floating-Point Binary Representation
-3.5	$(11.1)_2$	$-(0.111)_2 \cdot 2^2$, that is, $M=(0.111)_2$ and $c=2$
$\frac{7}{8}$	$(0.111)_2$	$(0.111)_2 \cdot 2^0$, that is, $M=(0.111)_2$ again, but $c=0$
$\frac{1}{16}$	$(0.0001)_2$	$(0.1)_2 \cdot 2^{-3}$, that is, $M=(0.1)_2$, and $c=-3$

คำว่า **Normalized** นั้นมาจากการที่ k-bit fraction M (ซึ่งเรียกว่า mantissa) มี $b_{-1} = 1$ ดังนั้น M >= 1/2

ส่วนคำว่า **floating point** นั้นอ้างถึงการที่เลขจำนวนเต็ม c (ซึ่งเรียกว่า characteristic หรือ exponent ของ X) ถูกปรับค่าเพื่อทำให้ binary point นั้นอยู่ทางซ้ายของ 1 (คือ b_{-1}) ใน binary representation ของ X (นั่นคือเพื่อทำให้ $b_{-1} = 1$ นั่นเอง)

เครื่องคำนวณแบบดิจิตัลทุกเครื่องจะเก็บ mantissa ในรูปที่ normalized แล้ว ทั้งนี้เพื่อให้แน่ใจว่าได้เก็บ bit ของ M มากที่สุดเท่าที่จะเป็นได้

ตัวอย่าง

$1/5 = 0.2 = (0.0011001100110011\ldots)_2$
นั้นอาจถูกเก็บใน 6-bit mantissa device ดังนี้

Not normalized: $(0.001100)_2 \cdot 2^0 = 1/8+1/16=3/16$

(error=1/5-3/16=1/80)

Normalized: $(0.110011)_2 \cdot 2^{-2} = (1/2+1/4+1/32+1/64) \cdot 1/4=51/256$

(error = 1/5-51/256 = 1/1280)

Normalized representation มีความแม่นยำมากกว่า Not normalized representation ถึง 16 เท่า ทั้งนี้เพราะไม่เสีย 2 bits เพื่อเก็บ (nonsignificant) leading zeros ของ M

Bit no.	31 30 29 28 27 26 25 . . . 9 8 7 6 5 4 3 2 1 0
	0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0
	23-bit for M → 7-bit for |c| →
	sign of X(+) bit ที่ไม่ปรากฏ คือ 0 sign of c(+)

รูปแสดงการเก็บ a floating point binary representation ใน a 32-bit word

จากรูป $c = +(0000011)_2 = +3$

$M = (0.11101)_2$ และ sign bit ของ X คือ 0 (สำหรับ +)

ดังนั้น $x = +(0.11101)_2 . 2^3 = +(111.01)_2 = 7\ 1/4$

1.3D <u>23-bit Mantissa Device สามารถเก็บเลขจำนวนขนาดใดได้บ้าง ?</u>

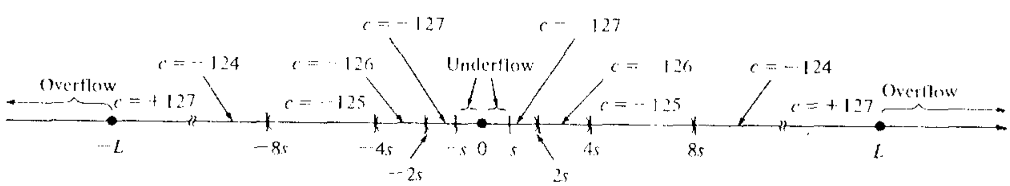
เราจะเรียก nonzero binary number ว่า X representable ถ้าเลขฐานสองจำนวนนั้นสามารถถูกเก็บได้ตามรูปบน นั่นคือถ้า

$X = \pm M.2^c$ โดยที่ M คือ normalized 23-bit fraction

และ $|c| <= 127$ ...(7)

$1/2 = (0.10 \ldots 0)_2 <= M <= (0.11\ldots1)_2 = 1- 2^{-23}$ (ทั้งนี้ b_{-1} ต้องเป็น 1)

ดังนั้น $(1/2)2^c <= M.2^c = |X| < 1.2^c$ (ทั้งนี้เพราะ $M < 1$)



รูป 1.3-1 ช่วงสำหรับ c ค่าต่างๆ กัน

และเนื่องจากว่า $-127 < c < 127$

และ $1/2 < M <= 1-2^{-23}$

ดังนั้น $s = (1/2)2^{-127} = 2^{-128} = 2.938736... \times 10^{-39}$ คือ smallest representable $X > 0$ ...(8)

และ $L = (1-2^{-23})2^{127} = 1.70141... \times 10^{38}$ คือ largest representable X ...(9)

ถ้าพยายามเก็บ x ซึ่ง $|x| > L$ แล้ว จะทำให้เกิด floating-point overflov

ในทำนองเดียวกัน ถ้าพยายามเก็บ x ซึ่ง $0 < |x| < s$ จะทำให้เกิด floating-point underflow

1.3E ความคลาดเคลื่อนแบบฝังติด (Inherent Error)

เลขจำนวนจริง x ส่วนมากจะไม่สามารถถูกเก็บอย่าง exactly ใน k-bit mantissa device เครื่องจะเก็บค่าประมาณ คือ $X = \pm M.2^{c}$ ซึ่งมี error ใน the least significant bit ของ M (นั่นคือ b_{-k}) เราเรียก error นี้ว่า **ความคลาดเคลื่อนแบบฝังติดในการเก็บค่า** x ซึ่งจะมีค่าน้อยกว่า 2^{-k}

ถ้า k = 23 แล้ว M จะมีความคลาดเคลื่อนน้อยกว่า $2^{-23} \approx 1.2 \times 10^{-7}$ ดังนั้น 23-bit mantissa device จะเก็บเลขจำนวนจริง x ได้แม่นยำถึงประมาณ 7 **ตัวเลขนัยสำคัญ** (7 significant digits)

ตัวอย่าง

$x = 0.1 = 0.8 \times 2^{-3}$, $x = M.2^{-3}$

$$M = (0.\overbrace{11001100110011001100110}^{23\text{ bits}})_2 = 6,710,886/8,388,608$$

$$= 0.7999999523...$$

ดังนั้น x = 0.1 จะเก็บอยู่ในค่าซึ่งเล็กกว่าค่าจริง

$x = (0.7999999523\ldots).(1/8) = 0.09999999404\ldots$

*To minimized inherent roundoff, variables x that are input as decimals should be entered rounded to as **many** significant digits as can possibly be stored on the device used.

จากตัวอย่างการวิ่งโปรแกรมในหัวข้อ 1.2A นั้น สำหรับเครื่องที่เก็บได้ 8 significant digits เราจะ enter ค่า $\pi = 3.1415927$

1.4 ความคลาดเคลื่อนในการคำนวณแบบ Fixed-Precision

1.4A Decimal-Place UP:: Significant-Digit Accuracy

"leading **significant** digit" ของเลขทศนิยมคือ "leftmost **nonzero** digit"

ดังนั้น $-0.\overbrace{00\underbrace{666667}_{6s}}^{8d}$ approximates $-1/150$

นั่นคือ $-1/150 \doteq -0.00666667(6s)$

หรือ $-1/150 \doteq -0.00666667(8d)$

$4\ 1/9 \doteq 4.011111\ldots = 4.01(2d$ หรือ $3s)$

$4\ 1/9 = 4.011(3d$ หรือ $4s)$

หรืออาจเขียน

$-1/150 \doteq -0.00666667$

$4\ 1/9 \doteq 4.01$ และ $4\ 1/9 \doteq 4.011$

Every digital device has a fixed precision that can be described in terms of the number of significant digits that it. can accurately store in one word.

ตัวอย่าง สำหรับ 23-bit mantissa device ที่สามารถเก็บเลขจำนวนได้ด้วยความแม่นยำประมาณ 7 ตัวเลขนัยสำคัญ นั่นคือ 7s device

real numbers: x, y, z,... (ใช้ lower case)

ค่า approximate: X, Y, Z,.. . (ใช้ upper case)

ดังนั้นใน 7s device: $x = X(7s)$

1.4B การคำนวณแบบ Fixed-Precision

พิจารณาเครื่องคิดเลขชนิด 4s, ฐาน 10

$X = \pm M.10^{c}$, $M = +0.d_1d_2d_3d_4$ $(d_1 \neq 0 \text{ if } X \neq 0)$

c เลขจำนวนเต็มซึ่งมีค่าระหว่าง -9 ถึง 9

(Normalized floating-point(decimal) representation)

ดังนั้น $x = -1/150$ จะถูกเก็บเป็น $X = -0.6667\text{x}10^{-2}$ $(M = 0.6667, c = -2)$

$y = 4\ 1/9$ จะถูกเก็บเป็น $Y = 0.4011\text{x}10^{1}$ $(M = 0.4011, c = +1)$

ค่าบวกที่เล็กที่สุดและใหญ่ที่สุดคือ $s = 0.1000 \times 10^{-9} = 10^{-10}$

และ $L = 0.9999 \times 10^{9} = 999{,}900{,}000$

ให้ "o" แทน +, -, *, /

ในการคำนวณหา $x \circ y$ จะหา $x \hat{\circ} Y$ แทน โดยที่

(**) $x \hat{\circ} Y$ หมายความว่า: หา $x \circ Y$ อย่างน้อย 5s แล้วปัดเศษให้เหลือ 4s

เราเรียกวิธีนี้ว่า "4s arithmetic" ซึ่งอาจเป็นผลให้เกิดความคลาดเคลื่อนหลายๆชนิด

1.4C ความคลาดเคลื่อนในการคำนวณแบบ Fixed-Precision

ban 5 จำนวนคือ

x=8846.4, y=0.0012495, z=0.40366, u-0.4037681, u=50 และจะถูกเก็บใน 4s device ดังนี้

$X=0.8846\times10^{4}$, $Y=0.1250\times10^{-2}$, $Z=0.4037\times10^{0}$, $W=0.4038\times10^{0}$

และ $U=0.5000\times10^{2}$

การคำนวณตาม (**) ในหัวข้อ 1.4B จะเกิด errors ต่างๆ เช่น

$x+y=8846.4012495 \neq X$, แต่ $X \hat{+} Y = 8846 = X$ (1)

$z+w=0.8074281 \doteq 0.8074$, แต่ $Z \hat{+} W = Z+W = 0.8075$ (2)

$z-y=0.4024105 \doteq 0.4024$, แต่ $Z \hat{-} Y = 0.4025$ (3)

$x^{2}=78,258,793(8s) \doteq 0.7826.10^{8}$, แต่ $X \hat{*} X = 0.7825.10^{8}$ (4)

$u/y=40016.0(6s) \doteq 0.4002.10^{5}$, แต่ $U \hat{/} Y = 0.4000.10^{5}$ (5)

$z-w=-0.0001081 \doteq -0.1081.10^{-3}$, แต่ $Z \hat{-} W = -0.0001 = -0.1000.10^{-3}$ (6)

(1)-(5) แสดงความคลาดเคลื่อนแพร่กระจายเนื่องจากการปัดเศษ (propagated roundoff error)

(1) แสดงการบวกเลขที่มีขนาดใหญ่มากๆ กับเลขที่มีขนาดเล็กมากๆ ซึ่งผลบวกก็คือเลขจำนวนที่มีขนาดใหญ่นั่นเอง

(2)-(5) แสดงว่า ถึงแม้ว่าเราจะปัดเศษ X และ Y ให้มีตัวเลขนัยสำคัญ s ตัว $X \hat{o} Y$ ก็อาจเกิดความคลาดเคลื่อนในตัวเลขนัยสำคัญตัวที่ s ได้ ดังนั้น digital device จึงมักเก็บตัวเลขไว้มากกว่าที่จะแสดงได้

(4)-(5) แสดงว่าความคลาดเคลื่อนจะถูกขยายขนาดขึ้นโดยการคูณด้วยจำนวนที่มีขนาดใหญ่ หรือโดยการหารด้วยจำนวนที่มีขนาดเล็ก

$(X \hat{*} X) - x^{2} = -8793$ แสดงว่าเกิดความไม่แม่นยำเล็กน้อยใน

$(U \hat{/} Y) - (u/y) = -16$ least significant digit แต่จริงๆ แล้ว ค่าความแตกต่างค่อนข้างใหญ่

เราเรียกความคลาดเคลื่อนนี้ว่าการขยายตัวของความคลาดเคลื่อน (error magnification) ซึ่งอาจจะเสียหายมากในบางสถานการณ์

(6) แสดงความคลาดเคลื่อนที่เกิดเมื่อทำการลบเลขสองจำนวนซึ่งมีค่าใกล้เคียงกัน ใน (6) เมื่อเอา W ไปลบจาก Z ตัวเลขนัยสำคัญสามตัวแรกถูกตัดทิ้งไป ตัวเลขนัยสำคัญที่เป็นตัวนำของผลลบ(คือของ Z-W) มาจากหลักที่ 4 ของ W และ Z เท่านั้น แสดงให้เห็นว่าเกิดความคลาดเคลื่อนมากเนื่องจากการปัดเศษ (roundoff) นั่นคือ loss of significance หรือ catastrophic cancellation

สรุปเกี่ยวกับความคลาดเคลื่อน

1) **Underflow/Overflow** การปฏิบัติการที่ทำให้ได้เลขจำนวนที่มีขนาดเล็กเกินไปหรือใหญ่เกินไปที่จะถูกเก็บในเครื่องได้

2) **ความคลาดเคลื่อนจากการปัดเศษ (Roundoff error)** เกิดเมื่อทำการคำนวณใน fixed-precision digital device ซึ่งเป็นผลมาจาก

   a) **ความคลาดเคลื่อนแบบฝังติด** (Inherent roundoff) คือการเก็บเลขฐานสองซึ่งเป็นเพียงค่าประมาณของเลขจำนวนจริง

   b) **ความคลาดเคลื่อนแพร่กระจาย** (**Propagate roundoff**) คือการใช้ fixed precision arithmetic กับเลขจำนวนซึ่งถูกเก็บไว้

      b.1) **Neligible addition** การบวกหรือลบเลขที่มีขนาดต่างกันมาก

      b.2) **การขยายตัวของความคลาดเคลื่อน (Error magnification)** การคูณด้วยเลขที่มีขนาดใหญ่หรือ การหารด้วยเลขที่มีขนาดเล็ก

      b.3) **การขาดนัยสำคัญ** (**Loss of significance**) การลบเลขสองจำนวนที่มีขนาดใกล้เคียงกันมาก (nearly equal)

1.4D ฟังก์ชันภายใน (Internal Functions)

บิลท์อินฟังก์ชัน (built-in function) เช่น sin, cos, exp, ln นั้นโดยทั่วๆไปแล้วจะทำให้เกิดความคลาดเคลื่อนแพร่กระจายซึ่งเนื่องมาจากการปัดเศษมากกว่าที่เกิดจากการคำนวณโดยใช้ตัวดำเนินการทางเลขคณิตตัวหนึ่งๆ (+, -, *, /)

ตัวอย่าง การคำนวณ x^z นั้นจะคำนวณโดยการใช้สูตร

$x^z = e^{z \ln x}$ ทั้งนี้โดยที่ฐาน x ต้องมีค่าเป็นบวก นั่นคือ

การคำนวณตามคำสั่งในภาษาต่างๆ เช่น

FORTRAN : X**Z

BASIC : X↑Z

เครื่องคิดเลข : x^z

ถ้า Z เป็นเลขจำนวนเต็มบวก (positive integer) แล้ว $x^z = e^{z \ln x}$ จะแม่นยำน้อยกว่า X*X*X*...*X (Z ครั้ง) สำหรับ Z ใดๆ (2 <= Z <= 4) ความคลาดเคลื่อนแพร่กระจายซึ่งเนื่องมาจากการปัดเศษจะลดลงได้โดยเขียน X*X*X*...*X แทน x^z ทั้งนี้นอกเสียจากว่า เราทราบว่าคอมพิวเตอร์ทำการคำนวณตามนี้โดยอัตโนมัติอยู่แล้ว

1.5 การหลีกเลี่ยงความคลาดเคลื่อนจากการปัดเศษ (Roundoff Error)

กฎทั่วๆไปในการหลีกเลี่ยงการเกิดความคลาดเคลื่อนหรืออย่างน้อยก็ทำให้เกิดน้อยที่สุด

1.5A กฎสำหรับการลดความคลาดเคลื่อนจากการปัดเศษ

กฎที่ 1 เพื่อทำให้ความคลาดเคลื่อนแบบฝังติดน้อยที่สุด จงใช้เลขจำนวนจริงโดยใช้ตัวเลขนัยสำคัญจำนวนเท่ากับที่เครื่องเก็บได้...(1.3E)

กฎที่ 2 เพื่อทำให้โอกาสของการเกิด overflow และ underflow น้อยที่สุด ให้พยายามเก็บค่าที่เกิดจากการคำนวณในทุกขั้นของการคำนวณให้ใกล้ ±1 (นั่นคือพยายามให้ characteristic มีค่าใกล้ศูนย์)

ตัวอย่าง ถ้าทราบขนาดของ x, y, z แล้ว

นิพจน์ xy/z ควรจะถูกคำนวณดังนี้

(xy)/z ถ้าระหว่าง $|x|$ หรือ $|y|$ มีตัวใดตัวหนึ่งใหญ่และอีกตัวเล็ก

x(y/z) ถ้า $|y|$ และ $|z|$ ทั้งสองตัวเล็ก หรือทั้งสองตัวใหญ่

(x/z)y ถ้า $|x|$ และ $|y|$ ทั้งสองตัวเล็ก หรือทั้งสองตัวใหญ่

ในเครื่องคอมพิวเตอร์ 32-bit word ส่วนมาก overflow (หรือ underflow) จะเกิดขึ้นเมื่อคำนวณ e^x สำหรับ x > 40 (x < -40) จึงทำให้จำเป็นและสำคัญมากที่จะต้อง **ให้ argument ของ exponential function นั้นใกล้ศูนย์มากเท่าที่จะทำได้**

ตัวอย่าง ถ้าทราบว่า x >> 0, z >> 0 และ n > 1

ในการคำนวณหรือเขียนคำสั่งในโปรแกรมควรจะทำดังนี้

หาค่าของ e^x/e^z ด้วยคำสั่งหา e^{x-z}

และหาค่าของ z^n/e^{nx} ด้วยคำสั่งหา $(z/e^x)^n$

กฎที่ 3 เพื่อช่วยทำให้ความคลาดเคลื่อนแพร่กระจายน้อยลง ให้ใช้นิพจน์สำหรับการคำนวณที่ต้องใช้การคำนวณทางเลขคณิตน้อยครั้งที่สุด

ตัวอย่าง เช่น $B = \sin^2 x / [x^2(1+\cos x)]$ (*)

ถ้าคำนวณโดยใช้ $(\sin x / x)^2/(1+\cos x)$ จะลดการคูณลงได้หนึ่งครั้งจากการคำนวณโดยตรงตามสูตร (*)

กฎข้อ 3 นี้จะใช้กับการคำนวณหาค่า polynomial ต่อไปนี้

Nested Multiplication Rule:

Polynomial:

Exponential form: $p_{exp}(x) = a_1x^5 + a_2x^4 + a_3x^3 + a_4x^2 + a_5x + a_6$

Nested form: $p_{nest}(x) = ((((a_1x + a_2)x + a_3)x + a_4)x + a_5)x + a_6$

$p_{exp}(x)$ นั้นการคำนวณต้องใช้ x^n subroutine ซึ่งมีความแม่นยำน้อยกว่า (less accurate) และถ้า x เป็นลบจะคำนวณไม่ได้

ตารางต่อไปนี้เปรียบเทียบความแม่นยำของ $p_{exp}(x)$ และ $p_{nest}(x)$
(ใช้ 23-bit mantissa device)

4th-degree polynomial

$$p_{exp}(x) = x^4 - 9.5x^3 + 28.49x^2 - 28.417x + 2.5662$$

$$p_{nest}(x) = (((x-9.5)x + 28.49)x - 28.417)x + 2.5662$$

$$p_{exp}(x) = p_{nest}(x) = (x-0.1)(x-2.1)(x-2.6)(x-4.7)$$

ตาราง 1.5-1 การเปรียบเทียบความแม่นยำของ $p_{exp}(x)$ และ $p_{nest}(x)$

x	Calculated $p_{exp}(x)$	Calculated $p_{nest}(x)$	Exact Value
2.4	0.317401'	0.317399	0.3174
2.5	0.211199	0.211200*	0.2112
2.7	-0.312001	-0.312000*	-0.3120
2.8	-0.718201	-0.718200*	-0.7182
2.9	-1 209599*	-1.209601	-1.2096
3.1	-2.399997	-2.400001'	-2.4000
3.2	-3.069001	-3.068999"	-3.0690
3.3	-3.763199	-3.763200*	-3.7632
3.5	-5.140801	-5.140800*	-5.1408
3.6	-5.775004	-5.775001'	-5.7750
3.7	-6.335999	-6.336001*	-6.3360
3.8	-6.793202	-6.793201*	-6.7932
3.9	-7.113601*	-7.113602	-7.1136
4.3	-6.283200*	-6.283197	-6.2832
4.4	-5.340597	-5.340602*	-5.3406
4.6	-2.250000*	-2.249999	-2.2500
4.7	0.000010	0.000001*	0.0000
4.8	2.791805	2.791802*	2.7918
4.9	6.182398	6.182401*	6.1824

* แสดงค่าที่แม่นยำกว่า ในตารางแสดงค่า x ซึ่งมีค่าระหว่าง 0.0(0.1)5.0
ซึ่ง $|p_{nest}(x) - p_{exp}(x)| > 10^{-6}$
รากของสมการ polynomial คือ 0.1, 2.1, 2.6 และ 4.7

จากตารางเราจะเห็นว่า

1. $p_{nest}(x)$ ส่วนมากแม่นยำเท่ากับ $p_{exp}(x)$

2. ทั้ง $p_{exp}(x)$ และ $p_{nest}(x)$ มีความคลาดเคลื่อนใกล้ๆรากตัวที่ใหญ่ที่สุดของสมการ polynomial (นั่นคือ x = 4.7)

โดยทั่วๆไป แม้ว่าจะใช้วิธี nested กับการหา nth-degree polynomial

$$p(x) = a_1x^n + a_2x^{n-1} + a_3x^{n-2} + \ldots + a_nx + a_{n+1} \quad (a_1 \neq 0)$$

ยังมีความลำบากที่จะให้ได้ความแม่นยำเมื่อค่าของ x อยู่ใกล้รากจริงที่มีขนาดใหญ่ โดยเฉพาะอย่างยิ่งถ้าสัมประสิทธิ์บางตัวมีขนาดใหญ่ด้วย (เป็นสาเหตุของการขยายตัวของความคลาดเคลื่อน) หรือ slope ใกล้ๆรากมีค่าเข้าใกล้ศูนย์ และยิ่ง n มีค่าสูงขึ้น ๆ สถานการณ์ก็จะยิ่งเลวลง

รูปต่อไปแสดง <u>Synthetic Division **Algorithm (Horner's method)**</u>
<u>**เพื่อหาค่า polynomial ในรูป nested**</u>

(a) **Algorithm: Synthetic Division (Horner's Method)**

Purpose: To evaluate $PolValue = a_1x^n + a_2x^{n-1} + \ldots + a_nx + a_{n+1}$

GET n, a_1, a_2, , a_{n+1}, x
Pol Value ← 0
DO FOR $i = 1$ TO $n + 1$
PolValue ← *PolValue* * x + a_i

(b) Find $p(x) = (((1x - 9.5)x + 28.49)x - 28.417)x + 2.5662$ when x = -2

$\boxed{-2}$		1	−9.5	28.49	−28.417	2.5662
	⊕		−2.0	23.0	−102.98	262.794
		1	−11.5	51.49	−131.397	$\boxed{265.3602} = p(-2)$

Note 1: Diagonal arrows indicate multiplication by x = -2.

Note 2: $\dfrac{p(x)}{x - (-2)} = x^3 - 11.5x^2 + 51.49x - 131.397 + \dfrac{265.3602}{x-(-2)}$,

รูป 1.5-1 (a) SYNTHETIC **Division Algorithm** สำหรับการหา p(x);
(b) p(-2); (c) POLVAL

(c)
```
      FUNCTION POLVAL(X, NP1, A)
      DIMENSION  A(NP1)
C - - - - - - - - - - - - - - - - - - - - - - - - C
C POLVAL  = A(1)*X**N + . . . + A(N)*X + A(NP1) C
C  WHERE   N  =  DEGREE  OF  POLYNOMIAL          C
C         NP1  =  NUMBER  OF  COEFFICIENTS  = N+1  C
C - - - - - - - - - - - - - - - - - - - - - - - - C
      POLVAL  =  0.0
      DO 1  I=1, NP1
    1    POLVAL  =  POLVAL*X  +  A(I)
      RETURN
      END
```

กฎที่ 4 ออกแบบโปรแกรมเพื่อหลีกเลี่ยงการขาดนัยสำคัญ (loss of significance) ที่อาจเกิดขึ้นเมื่อทำการบวกหรือการลบ

ตัวอย่างใน 1.5B-1.5D จะแสดงการใช้กฎข้อ 4 นี้

1.5B การจัดนิพจน์ตรีโกณมิติเสียใหม่ (Trigonometric Rearrangement) เพื่อหลีกเลี่ยงการขาดนัยสำคัญ (Loss of Significance)

ปัญหา

อธิบายการหาค่า $f(x) = (1-\cos x)/x^2$ อย่างถูกต้องแม่นยำ สำหรับทุกค่าของ $x \neq 0$

Solution

จาก 1.2A การคำนวณค่าของ $A = (1-\cos x)/x^2$ นั้นค่าจะไม่แม่นยำเมื่อ $x \approx 0$ เหตุผลเพราะ ($\cos x \approx 1$) เมื่อ ($x \approx 0$) เศษจะเกิดการขาดนัยสำคัญ และเมื่อหารด้วย x^2 ซึ่งมีค่าน้อยความคลาดเคลื่อนจะมีมากขึ้น ส่วนการหาค่าของ $B = (\sin x/x)^2[1/(1+\cos x)]$ จะไม่แม่นยำเมื่อ $x \approx \pi$ ทั้งนี้เพราะ ($\cos x \approx -1$) เมื่อ ($x \approx \pi$) ทำให้ตัวหารขาดนัยสำคัญและค่าของ B ก็จะเล็กลง (คำตอบจะดู absurd)

ดังนั้นเมื่อเข้าใจปัญหาที่เกิดขึ้น เราอาจหลีกเลี่ยงสิ่งที่จะเกิดโดยการคำนวณ f(x) ดังนี้

$$f(x) = \begin{cases} B & \text{ถ้า } (1-\cos x) \approx 0 \quad (|1-\cos x| <= 0.01) \\ A & \text{ถ้า } |1-\cos x| > 0.01 \end{cases}$$

1.5C การจัดนิพจน์พีชคณิตเสียใหม่ (Algebraic Rearrangement) เพื่อหลีกเลี่ยงการขาดนัยสำคัญ

จาก 1.2B เราพบว่า ถ้า $b_1^2 >> |c_1|$ และ $b_1^2 - c_1$ จะไม่ต่างจาก b_1^2 มากดังนั้น $r(-) = b_1 - \sqrt{b_1^2 - c_1}$ จะ loss of significance ถ้า $b_1 > 0$

$r(+) = b_1 + \sqrt{b_1^2 - c_1}$ จะ loss of significance ถ้า $b_1 < 0$

ถ้าหากจะหารากของ $ax^2 + bx + c = 0$ $(a \neq 0)$ โดยที่ไม่ให้ขาดนัยสำคัญ โดยใช้สูตร

$$root_1 = (\pm)\left(|b_1| + \sqrt{b_1^2 - c_1}\right) \text{ และ}$$

$$root_2 = c_1/root_1$$

โดยที่ $b_1 = -b/2a$, $c_1 = c/d$ และ $(\pm)$ คือ $(+)$ นอกจาก $b_1 < 0$

ต่อไปนี้เป็นโปรแกรมย่อยซับรูทีนในภาษา FORTRAN เพื่อหารากของสมการกำลังสอง $ax^2 + bx + c = 0$, $a \neq 0$ ตามวิธีที่กล่าวข้างต้น

```
      SUBROUTINE QROOTS(A, B, C, ROOTl, ROOT2, COMPLX, IY, PRINT)
      LOGICAL PRINT, COMPLX
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - C
C THIS SUBROUTINE FINDS THE TWO ROOTS OF THE QUADRATIC          C
C            AX**2 + BX + C                                     C
C IF PRINT = TRUE, IT PRINTS THEM ON OUTPUT DEVICE IW.          C
C REAL ROOTS (COMPLX=FALSE) ARE RETURNED AS ROOT1 AND ROOT2,    C
C AND COMPLEX ROOTS (COMPLX=TRUE) AS ROOT1 +OR- I*ROOT2.        C
C - - - - - - - - - - - - - - - - VERSION , 5/1/81 - - - - - -  C
      B1 = -0.5*B/A
      Cl = CIA
      DISCR = B1*B1 - Cl
      IF (DISCR .LT. 0.) GOT0 10
C
C     REAL ROOTS: ROOT1 AND ROOT2
         COMPLX = .FALSE.
         ROOT1 = ABS(B1) + SQRT(DISCR)
         IF (B1 .LT. 0.) ROOT1 = -ROOT1
         ROOT2 = 0.0
         IF (ROOT1 .NE. 0.) ROOT2 = C1/ROOT1
C
         IF (PRINT) WRITE(IW,1) ROOTl, ROOT2
    1    FORMAT(' REAL ROOTS: ',E14.7,' AND ',E14.7)
         RETURN
C
C     COMPLEX CONJUGATE ROOTS: ROOT1 +OR- I*ROOT2
   10    COMPLX = .TRUE.
         ROOT1 = B1
         ROOT2 = SQRT(-DISCR)
C
         IF (PRINT) WRITE(IW,2) ROOT1, ROOT2
    2    FORMAT(' COMPLEX ROOTS:',E15.7,' +OR- I*(',E14.7,')')
         RETURN
C
      END
```

รูป 1.5-2 ซับรูทีนภาษาฟอร์แทรนชื่อ QROOTS สำหรับหารากของสมการกำลังสอง

โปรแกรมหลักเพื่อหารากจริงของสมการ $x^2 \pm 100.01x + 1 = 0$ และรากเชิงซ้อน (complex roots) ของสมการ $x^2 - 2x + 3 = 0$ การคำนวณแม่นยำอย่างน้อยที่สุด 7 ตัวเลขนัยสำคัญ (7s)

```
C * * * * * * * * * TESTQR  *  * * * * * * * * * * * * *
C INTERACTIVE PROGRAM TO TEST SUBROUTINE QROOTS (IW = 5)
     LOGICAL  COMPLX
C
     WRITE (5,1)
   1 FORMAT('OTO FIND ROOTS OF AX**2+BX+C, INPUT A, B, C')
     READ (5,*) A, B, C
C
     CALL QROOTS (A, B, C, RI, R2, COMPLX, 5, .TRUE.)
C
     STOP
     END
```

```
TO FIND ROOTS OF AX**2+BX+C, INPUT A, 8, C
1,  -100.01,  1

REAL  ROOTS:  0.1000000E+03 AND  0.1000000E-01


TO FIND ROOTS OF AX**2+BX+C, INPUT A, B, C
1, +100.01, 1

REAL  ROOTS:  -0.1000000E+03 AND  -0.1000000E-01


TO FIND ROOTS OF AX**2+BX+C, INPUT A, B, C
1,  -2,  3

COMPLEX  ROOTS:  0.1000000E+01  +OR-  I*( 0.1414214E+01)
```

รูป 1.5-3 โปรแกรมซึ่งเรียกใช้ QROOTS

1.5D การใช้อนุกรมของแมคคลอลินเพื่อหลีกเลี่ยงการขาดนัยสำคัญ

ปัญหา คำนวณ $f(x) = (e^x - 1)/x$ อย่างถูกต้องแม่นยำสำหรับทุก $x \neq 0$

Solution เพราะว่า $e^x \dashrightarrow 1$ ขณะที่ $x \dashrightarrow 0$ เศษของ $f(x)$ จะขาดนัยสำคัญและจะขยายตัวมากขึ้น (magnified) โดยตัวส่วนที่มีขนาดเล็ก เมื่อ $x \approx 0$ ปัญหานี้ต่างไปจาก 1.5B เพราะเราไม่อาจหา equivalent expression เพื่อช่วยหลีกเลี่ยงปัญหาข้างต้น ในสถานการณ์เช่นนี้เราจะใช้อนุกรมของแมคคลอลินที่เหมาะสมเข้าช่วย

อนุกรมของแมคคลอลินสำหรับ

e^x คือ $[1 + x + x^2/2! + x^3/3! + R_3(x)]$

$f(x) = (1/x)\{[1+x+x^2/2!+x^3/3!+R_3(x)]-1\}=1+x/2!+x^2/3!+R_3(x)/x$

โดยที่ $R_3(x)$ คือ ***แบบลากรองจ์ของเศษ*** *(Lagrange for8 of **the** remainder)*

$R_3(x) = [\exp(\lambda)/4!]\ x^4$ โดยที่ λ อยู่ระหว่าง 0 และ x

โดยที่ $|x| < 0.01$ แล้ว $R_3(x)/x$ (ซึ่งคือความคลาดเคลื่อนจากการตัดปลาย (truncation error) ของการประมาณ f(x) ด้วย $1 + x/2! + x^2/3!$) จะสอดคล้องกับ

$$\left|R_3(x)/x\right| = [\exp(\lambda)\left|x\right|^3]/24 < \exp(0.01)(0.01)^3/24 < 0.00000005$$

สิ่งนี้รับประกันว่า $1 + x/2! + x^2/3!$ ประมาณ $[\exp(x)-1]/x$ ถึง 7d ถ้า $|x| < 0.01$

ดังนั้นถ้าเราหาค่า f(x) ดังนี้

$$f(x) = \begin{cases} [\exp(x) - 1]/x & \text{ถ้า } |x| >= 0.01 \\ 1+(1/2)x[1+(1/3)x] & \text{ถ้า } |x| < 0.01 \end{cases}$$

แล้ว f(x) จะถูกคำนวณที่ความแม่นยำของเครื่องเมื่อ $|x| >= 0.01$ และด้วยความแม่นยำถึง 7d เมื่อ $|x| < 0.01$

1.5E Extended Precision

การขยายที่ที่ใช้เก็บเลขจำนวนจาก 1 เวิร์ด (word) เป็น 2 เวิร์ด นั้นทำให้สามารถเก็บตัวเลขนัยสำคัญได้มากกว่าเดิม ทั้งนี้เพราะเวิร์ดที่ถูกเพิ่มเข้าไปนั้นจะใช้ขยายส่วนที่เก็บ mantissa จึงทำให้เก็บตัวเลขนัยสำคัญได้อย่างน้อยสองเท่าของขนาดปกติ

ตัวอย่าง FORTRAN: กำหนดที่ขนาด 2 เวิร์ด ด้วยคำสั่ง

```
DOUBLE PRECISION variable list
```

การคำนวณโดยความเที่ยงตรงสองชั้น (double precision) ใช้เวลาเป็นสองเท่าของการคำนวณโดยความเที่ยงตรงชั้นเดียว (single precision) ดังนั้นจึงควรใช้กฎที่ 1-4 ก่อนเพื่อทำให้การใช้การคำนวณโดยความเที่ยงตรงชั้นเดียวมีประสิทธิภาพ นอกจากว่าต้องการความแม่นยำสูง และไม่สามารถใช้กฎทั้ง 4 ได้

1.6 ความคลาดเคลื่อน ความแม่นยำ และ การทดสอบเพื่อตรวจความใกล้เคียง

ในหัวข้อนี้จะศึกษาเทอมต่างๆ ที่ใช้ และหามาตรการเพื่อทำให้ได้ความแม่นยำเท่าที่ต้องการ

1.6A ความคลาดเคลื่อนสัมบูรณ์ (Absolute Error) ความคลาดเคลื่อนสัมพัทธ์ (Relative Error) และ ความคลาดเคลื่อนสัมพัทธ์คิดเป็นร้อยละ (Percent Error)

X เป็นค่าประมาณของค่าจริง x

ความแตกต่าง $\epsilon_x = x - X =$ (exact value) - (approximate value)

คือ ความคลาดเคลื่อนสัมบูรณ์ (absolute error หรือ error) (บางเล่มอาจใช้ $\epsilon_x = X - x$) ดังนั้น $x = X + \epsilon_x$

ถ้า $x \neq 0$ แล้วอัตราส่วน

$\rho_x = \epsilon_x / x = (x-X)/x$ **คือ ความคลาดเคลื่อนสัมพัทธ์** (relative error) สำหรับการประมาณ x ด้วย X

ในสถานการณ์ที่ทราบค่า ρ_x แล้วความคลาดเคลื่อนสัมบูรณ์อาจหาได้โดย

$\epsilon_x = X\rho_x$

ความคลาดเคลื่อนสัมพัทธ์ ρ_x บอกว่าส่วนใดของ x ยังผิดพลาดอยู่

ตัวอย่าง $\left|\rho_x\right| = 0.02$

$$\left|\epsilon_x\right| < 0.02 \left|x\right|$$

หรือกล่าวว่า X ประมาณค่า x ด้วยความคลาดเคลื่อนไม่เกิน 2 %

โดยทั่วๆไป $100\rho_x = 100\ \epsilon_x/x$ คือ **ความคลาดเคลื่อนสัมพัทธ์คิดเป็นร้อยละ** (percent error) ของการประมาณ x ด้วย X

ตัวอย่าง ถ้า $x = 1/30$ ถูกประมาณค่าด้วย $x = 0.033(2s)$ ดังนั้น

$$\epsilon_x = 1/30 - 0.033 = 1/30 - 99/3000 = 1/3000 \text{ และ}$$

$$\rho_x = (1/3000)/(1/30) = 1/100 = .01$$

ดังนั้น 0.033 ประมาณ 1/30 ด้วยความคลาดเคลื่อนไม่เกิน 1%

1.6B <u>ความเกี่ยวข้องกันของ ϵ_x และ ρ_x ต่อความแม่นยำของ X</u>

ถ้า $X = 2.3579$ --> $0.2358(4s)$

หรือ $X = 2.357682$ --> $0.2358(4s)$

เราต้องหาวิธีบอกคอมพิวเตอร์ให้เลือก X ที่ตรงกับความแม่นยำที่เราต้องการ

<u>การทดสอบความคลาดเคลื่อนสัมบูรณ์ (Absolute Error Test)</u>

d = เลขจำนวนเต็มบวกใด ๆ

$$|\epsilon_x| = |x - X| < (1/2).10^{-d}$$ --> จะประกันว่า X ประมาณค่า x ได้แม่นยำถึงทศนิยมตำแหน่งที่ d

ถ้า $x \neq 0$ แล้วความคลาดเคลื่อนสัมพัทธ์ $\rho_x = \epsilon_x/x$ สามารถถูกใช้เพื่อให้ได้ความแม่นยำเป็นจำนวนตัวเลขนัยสำคัญที่ต้องการ

<u>การทดสอบความคลาดเคลื่อนสัมพัทธ์ (Relative Error Test)</u>

s = เลขจำนวนเต็มบวกใด ๆ

(1) $|\rho_x| < (1/2).10^{-s}$ --> จะประกันว่า x ประมาณค่า x

(2) หรือ $|x - x_I| < (1/2).10^{-s}|x|$ ได้แม่นยำถึง s ตัวเลขนัยสำคัญ

<u>พิสูจน์</u> จาก $x = \pm M.10^c$

โดยที่ $0.1 <= M = 0.d_1 d_{-2} d_{-3} \ldots d_{-s} \ldots < 1$ ...(3)

ถ้า (1) และ (2) จริงแล้ว จาก (2) และ (3)

$$\left|\epsilon_x\right| = \left| x-X \right| < (1/2).10^{-s} \left| x \right| = ((1/2).10^{-s})(M.10^{c}) < ((1/2).10^{-s})10^{c}$$

แสดงว่า **ขนาดของความคลาดเคลื่อนของ** X < 1/2 ณ ทศนิยมตำแหน่งที่ s ของ M ใน (3) (นั่นคือ X มีความแม่นยำถึงจำนวนตัวเลขนัยสำคัญ = s ตัว)

1.7 ขั้นตอนวิธีของการทำซ้ำ (Iterative Algorithms)

ทุกขั้นตอนวิธีของการทำซ้ำสำหรับหาค่าที่ต้องการ ($\bar{x}$) นั้นจะทำ 2 ขั้นตอนต่อไปนี้

(1) ขั้นเริ่มต้น: หา **ค่าเดาเริ่มต้นค่าหนึ่ง** (initial guess) คือ x_o ซึ่งจะประมาณค่า $\bar{x}$

(2) ขั้นทำซ้ำ : สำหรับ k = 0, 1, 2,... ไปจนกระทั่งการทดสอบเพื่อหยุดการทำซ้ำเป็นจริง

ใช้ x_k เพื่อหา x_{k+1} ซึ่งเป็นค่าประมาณที่ถูกปรับปรุงแล้ว

เราเรียก x_k ว่า **การทำซ้ำครั้งที่ k** (kth iterate) และให้

$\epsilon_k = \bar{x} - x_k$ = **ความคลาดเคลื่อนของ** x_k ดังนั้น ϵ_k คือค่าที่จะบวกกับ x_k เพื่อให้ได้ $\bar{x}$ ($\bar{x} = x_k + \epsilon_k$)

ขั้นตอนวิธีจะสำเร็จ (success) ถ้า $x_k \to \bar{x}$ (นั่นคือ $\epsilon_x \to 0$) เมื่อ $k \to \infty$

ขั้นตอนวิธีจะแสดงลักษณะต่อไปนี้

1) Robustness

x_k จะมีค่าลู่เข้า $\bar{x}$ ถึงแม้ว่า x_o จะไม่ใช่ค่าเดาเริ่มต้นที่ดี

2) Rapid Convergence

เมื่อ x_k เข้าใกล้ $\bar{x}$, x_{k+1} จะเข้าใกล้มากกว่า

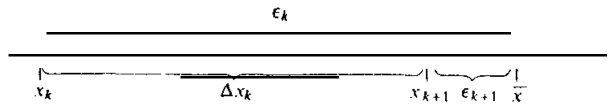
นั่นคือ $\left|\epsilon_{k+1}\right| = \left|\bar{x} - x_{k+1}\right| << \left|\bar{x} - x_k\right| = \left|\epsilon_k\right|$

โดยที่ $\left|\epsilon_{k+1}\right| << \left|\epsilon_k\right|$

ดังนั้น

$\Delta x_k = x_{k+1} - x_k \approx \bar{x} - x_k \simeq \epsilon_k$ (ความคลาดเคลื่อนที่แท้จริงซึ่งไม่ทราบค่า)

คือ increment จาก x_k ถึง x_{k+1}

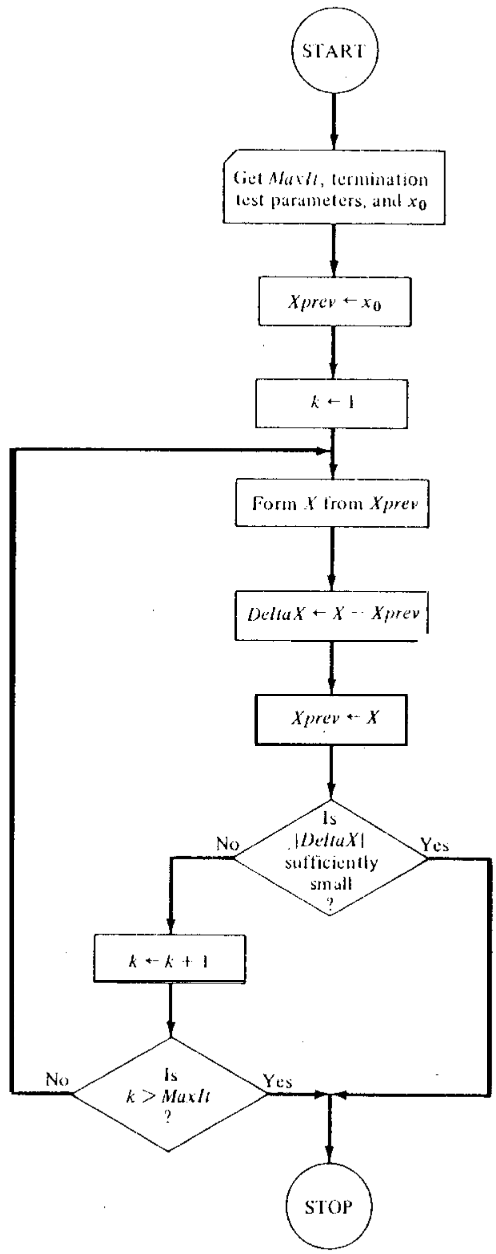


รูป 1.7-1 รูปแสดง Δx_k, ϵ_k, และ ϵ_{k+1}

Δx_k เป็นค่าประมาณที่คำนวณได้ของ ϵ_k

การทดสอบเพื่อหยุดการทำซ้ำ (Termination test) ส่วนมากสำหรับขั้นตอนวิธีของการทำซ้ำซึ่งรวมการหยุดงาน (และใช้ $x_{k+1} \approx \bar{x}$) เมื่อ Δx_k มีค่าเล็กมากพอ

1.7A ผังโปรแกรมสำหรับขั้นตอนวิธีของการทำซ้ำ



รูป 1.7-2 ผังภาพสำหรับ General Iterative Algorithm

General Iterative Algorithm

Purpose: To find $\bar{x}$ to a desired accuracy by repeatedly forming an improved approximation X from a current approximation ***Xprev*** until either X is sufficiently **close** to Xprev **(termination test)** or ***MaxIt*** iterations occur.

(initialize)
GET *MaxIt*, termination parameters, x_0 [initial guess)
Xprev $\leftarrow x_0$

{***iterate***}
DO FOR $k = 1$ **TO** *MaxIt* **UNTIL termination test** is satisfied
  BEGIN
  (form X) X $\leftarrow$ (formula involving Xprev)
  DeltaX $\leftarrow$ ***X*** **–** ***Xprev***   {*DeltaX* is $\Delta x_{k-1} = x_k - x_{k-1}$}
  [update) Xprev $\leftarrow$ ***X***   (ready for next iteration]
  (termination test: $|DeltaX| \approx 0$, as specified by termination parameters]
  END

(If **termination test** succeeded, then $\bar{x} \doteq X$ to the desired accuracy. If not, ***MaxIt*** iterations failed to yield an X close enough to $\bar{x}$.}

รูป 1.7-3 รหัสเทียมสำหรับ **General Iterative Algorithm**

1.7B การทดสอบเพื่อหยุดการทำซ้ำ (Termination Tests) สำหรับขั้นตอนวิธีของการทำซ้ำ

จากการประมาณค่า: $\Delta x_k \approx \epsilon_k$ และ ความคลาดเคลื่อนสมบูรณ์และความคลาดเคลื่อนสัมพัทธ์ (ในข้อ1.6B) ทำให้เราได้การทดสอบเพื่อหยุดการทำซ้ำ ดังนี้

Absolute Difference Test(For NumDec decimal-place accuracy)

ถ้า $|\text{DeltaX}|$ <= AbsTol โดยที่ AbsTol = Const * $10^{-\text{NumDec}}$
แล้ว X จะประมาณ $\bar{x}$ ได้แม่นยำถึงทศนิยมตำแหน่งที่ NumDec

Relative Difference **Test**(For **NumSig** significant-digit Accuracy)
ถ้า $\lvert DelTaX \rvert <= RelTol* \lvert X \rvert$ โดยที่ $RelTol = Const*10^{-NumSig}$ แล้ว X จะประมาณ $\bar{x}$ ได้แม่นยำถึงตัวเลขนัยสำคัญเท่ากับ NumSig ตัว

$$\boxed{DelTaX = X - X_{Prev}}$$

Const เรามักใช้ค่า 1 ถ้าการลู่เข้า (convergence) ช้า จะเลือกค่าเข้าใกล้ 0.5 และถ้าคาดว่ามีความคลาดเคลื่อนจากการปัดเศษมาก จะเลือกค่ามากกว่า 1 (คือ 2-9)

ถ้าเราต้องการความแม่นยำถึง 5 ตัวเลขนัยสำคัญ $\bar{x} \approx 3000$ เราควรใช้ NumDec = 1 ใน Absolute Difference Test แต่ถ้าทราบว่า $\bar{x} \approx -0.003$ เราควรใช้ NumDec = 7 ส่วนใน Relative Difference Test เราจะใช้ NumSig = 5 ในทั้ง 2 กรณี นั่นแสดงว่า Relative Difference Test มักจะถูกใช้ในโปรแกรมซึ่งจะหาค่า $\bar{x}$'s ที่มีขนาดต่างๆกันไป

อย่างไรก็ดี Relative Difference Test นั้นไม่ใช้กับการทดสอบการเข้าใกล้ศูนย์ อันที่จริงแล้ว ถ้า $\bar{x} = 0$ และ ขั้นตอนวิธีลู่เข้าเร็ว (i.e.,x_n --> 0 อย่างเร็ว) แล้ว $\lvert X \rvert << \lvert X_{Prev} \rvert$ ผลที่ตามมาคือ ถ้าไม่มีความคลาดเคลื่อนจากการปัดเศษแล้ว $\lvert DelTaX \rvert \approx \lvert X_{Prev} \rvert$ และอสมการใน Relative Difference Test จะไม่เป็นจริง

คอมพิวเตอร์อาจพบ ลูปอนันต์ (infinite loop) อย่างที่เราไม่คาดฝัน เราควรจะป้องกันโดยยึดหลักต่อไปนี้

For **any** iterative algorithm, no matter hov sure **you** are that a termination test will be set, put an upper limit (e.g., MaxIt) on the number of iterations (just in case you are wrong).

1.7C การลู่เข้าแบบเชิงเส้น (Linear Convergence) และ การลู่เข้าแบบอันดับสอง (Quadratic Convergence)

(*) $\boxed{(Dx/\text{preceding } Dx) \approx \text{Constant} = C}$ นั่นคือ $\Delta x_k \approx C\Delta x_{k-1}$

ถ้าเป็นไปตาม (*) นั่นคือเมื่อ $(x_{k+1}-x_k)$ is (approximately) proportional to (x_k-x_{k-1}) เราเรียกการลู่เข้าของ x_k to $\bar{x}$ ว่า การลู่เข้าแบบเชิงเส้น **(linear convergence)**

เหตุการณ์ข้างต้นตรงข้ามกับเมื่อการทำซ้ำ satisfied
$(Dx/\text{preceding } Dx) \to 0$ เมื่อ k เพิ่มขึ้น

(**) $\boxed{Dx/(\text{preceding } Dx)^2 \approx \text{Constant} = C}$ นั่นคือ $\Delta x_k \approx C(\Delta x_{k-1})^2$

ถ้าเป็นไปตาม (**) นั่นคือเมื่อ $(x_{k+1}-x_k)$ is (approximately) proportional to $(x_k-x_{k-1})^2$ เราเรียกการลู่เข้าของ x_k to $\bar{x}$ ว่า การลู่เข้าแบบอันดับสอง **(quadratic convergence)**

เราจะใช้ การลู่เข้าแบบอันดับสอง เพื่อเป็นมาตรฐานของเราสำหรับการลู่เข้าอย่างเร็ว (rapid convergence)

และจะใช้ การลู่เข้าแบบเชิงเส้น เพื่อเป็นมาตรฐานของเราสำหรับการลู่เข้าอย่างช้า (slow convergence)