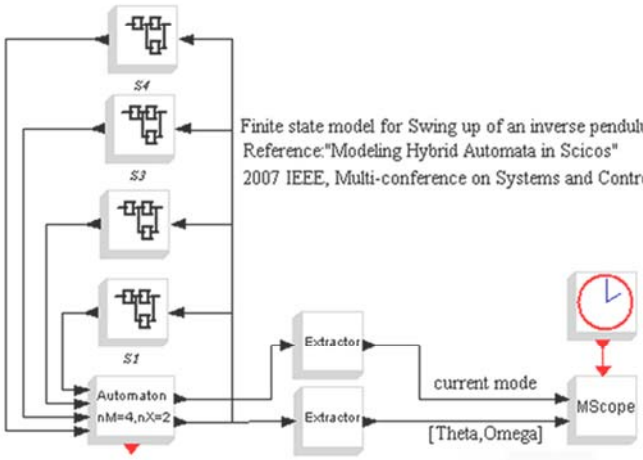
คู่มือโปรแกรมภาษา SCILAB

สำหรับผู้เริ่มต้น

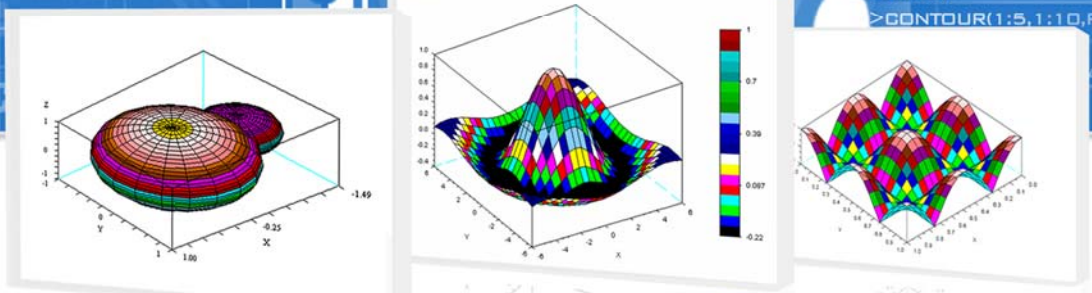
(ฉบับปรับปรุงใหม่)



Finite state model for Swing up of an inverse pendulum
Reference:"Modeling Hybrid Automata in Scicos"
2007 IEEE, Multi-conference on Systems and Control, Singapore

```
->//MULTIPLE POSTSCRIPT FILES FOR LA
->DRIVER('POS')
->T=%PI*(-10:10)/10;
->PLOT3D1(T,T,SIN(T)'*COS(T),THETA=
->XEND()
->CONTOUR(1:5,1:10,RAND(5,10),5);
->XEND()
->CHAMP(1:10,1:10,RAND(10,10),RAND
->XEND()
->T=%PI*(-10:10)/10;
->FUNCTION Z=SURF(X,Y),Z=SIN(X
->RECT=[-%PI,-%PI,%PI,%PI],-5,1
->Z=FEVAL(T,T,SURF)
->CONTOUR(T,T,Z,10,
```

```
->U = LINSPACE(-%PI/2, %PI/2, 20);
->V = LINSPACE(0, 2*%PI, 10);
->X = COS(U)'*COS(V);
->Y = COS(U)'*SIN(V);
->Z = SIN(U)'*ONES(V);
->SUBPLOT(1, 2, 1); PLOT3D3(X, Y, Z);
->SUBPLOT(1, 2, 2); SURF(X, Y, Z);
```



ผศ.ดร.ปิยะ โควินท์ทวีวัฒน์

คู่มือโปรแกรมภาษา SCILAB

สำหรับผู้เริ่มต้น

(ฉบับปรับปรุงใหม่)

ผศ.ดร.ปิยะ โควินท์ทวีวัฒน์

โปรแกรมวิศวกรรมโทรคมนาคม

มหาวิทยาลัยราชภัฏนครปฐม

คู่มือโปรแกรมภาษา SCILAB สำหรับผู้เริ่มต้น

(ฉบับปรับปรุงใหม่)

ข้อมูลทางบรรณานุกรมของหอสมุดแห่งชาติ

ปิยะ โควินท์ทวีวัฒน์

คู่มือโปรแกรมภาษา SCILAB สำหรับผู้เริ่มต้น./ ปิยะ โควินท์ทวีวัฒน์. --นครปฐม:
โปรแกรมวิศวกรรมโทรคมนาคม คณะวิทยาศาสตร์และเทคโนโลยี
มหาวิทยาลัยราชภัฏนครปฐม, 2551
310 หน้า
1. ภาษาคอมพิวเตอร์ 2. เอสซีไอแอลเอบี (ภาษาคอมพิวเตอร์). 3. ชื่อเรื่อง
GA 76.73.S3ป621
ISBN 974-620-609-5

หมายเหตุ

- หนังสืออิเล็กทรอนิกส์ฉบับนี้ได้ถูกจัดทำขึ้นเพื่อใช้ในการศึกษาและการทำวิจัย ผู้อ่านสามารถดาวน์โหลดไปใช้งานได้โดยไม่ต้องเสียค่าใช้จ่ายใดๆ
- ขอขอบคุณสำนักงานคณะกรรมการวิจัยแห่งชาติ (วช.) ที่ให้ทุนสนับสนุน

สำหรับครอบครัว โควินท์ทวีวัฒน์

คำนำ

ในปัจจุบันนี้ความเจริญก้าวหน้าทางด้านเทคโนโลยีเป็นไปอย่างรวดเร็ว หลายๆ หน่วยงานในภาคการศึกษาได้มีการนำอุปกรณ์คอมพิวเตอร์และซอฟต์แวร์ต่างๆ มาใช้ช่วยในการเรียนการสอนและทำงานวิจัยในหลายสาขาวิชาเช่น วิศวกรรมศาสตร์ และวิทยาศาสตร์ เป็นต้น โปรแกรม MATLAB ถือว่าเป็นซอฟต์แวร์ที่นิยมใช้งานกันมาก สำหรับการคำนวณเชิงตัวเลขและการแสดงผลกราฟิกที่ซับซ้อน โดยเฉพาะอย่างยิ่งสำหรับผู้ที่ทำงานทางด้านวิศวกรรมและวิทยาศาสตร์ ทั้งนี้เป็นเพราะว่าโปรแกรม MATLAB ง่ายต่อการเรียนรู้และทำความเข้าใจ อย่างไรก็ตามโปรแกรม MATLAB เป็นโปรแกรมที่ต้องเสียเงินค่าลิขสิทธิ์ของซอฟต์แวร์ซึ่งมีราคาแพงมาก ทำให้ผู้คนจำนวนมากเสียโอกาสที่จะศึกษาและเรียนรู้การใช้งานโปรแกรมนี้ เพื่อนำมาช่วยในการทำงานของตนเอง

โปรแกรม SCILAB เป็นโปรแกรมที่พัฒนาโดยกลุ่มของนักวิจัยจาก INRIA และ ENPC ในประเทศฝรั่งเศส เมื่อปี ค.ศ. 1990 โดยมีจุดมุ่งหมายเพื่อใช้ในการคำนวณเชิงตัวเลขและแสดงผลกราฟิกที่ซับซ้อนเช่นเดียวกับโปรแกรม MATLAB แต่โปรแกรม SCILAB เป็นโปรแกรมที่ให้ฟรี (ไม่เสียเงินค่าลิขสิทธิ์ของซอฟต์แวร์) ผู้ที่สนใจสามารถดาวน์โหลดโปรแกรมนี้ไปใช้งานได้ทันทีจากเว็บไซต์ http://home.npru.ac.th/~t3058/Scilab.html หรือ http://www.scilab.org จากที่ข้าพเจ้าได้ศึกษาการใช้งานโปรแกรม SCILAB พบว่าเป็นโปรแกรมที่สามารถทำงานได้อย่างมีประสิทธิภาพใกล้เคียงกับโปรแกรม MATLAB ในหลายๆ ด้าน ดังนั้นจะเห็นได้ว่าในปัจจุบันนี้มหาวิทยาลัยหลายแห่งทั้งในประเทศและต่างประเทศได้เริ่มมีการนำโปรแกรม SCILAB มาใช้ช่วยในการเรียนการสอนและทำงานวิจัย

หนังสือ “คู่มือโปรแกรมภาษา SCILAB สำหรับผู้เริ่มต้น” มีจุดมุ่งหมายเพื่ออธิบายถึงพื้นฐานการใช้งานโปรแกรม SCILAB ในรูปแบบที่ง่ายต่อการเข้าใจและเรียนรู้ได้ด้วยตนเอง ทำให้สามารถนำโปรแกรมนี้ไปประยุกต์ใช้ในการเรียนการสอนของวิชาต่างๆ ทางด้านวิศวกรรมและ

วิทยาศาสตร์ได้อย่างรวดเร็ว ซึ่งจะช่วยทำให้นักศึกษาสามารถเข้าใจถึงทฤษฎีในบทเรียนมากยิ่งขึ้น ฉะนั้นหนังสือเล่มนี้จึงสามารถนำมาใช้ประกอบการเรียนการสอนในระดับมัธยมศึกษาตอนปลาย และระดับอุดมศึกษาได้เป็นอย่างดี เช่น วิชาคณิตศาสตร์, วิชาฟิสิกส์, วิชาการสื่อสารดิจิทัล (digital communication), วิชาสัญญาณและระบบ (signals and systems), และวิชาการประมวลผลสัญญาณดิจิทัล (digital signal processing) เป็นต้น

หนังสือเล่มนี้จะเริ่มต้นอธิบายจาก ความรู้ทั่วไปเกี่ยวกับโปรแกรม SCILAB เช่น ประวัติความเป็นมาของโปรแกรม SCILAB และการใช้งานคำสั่งพื้นฐานทั่วไป เป็นต้น บทที่ 2 จะกล่าวถึงประเภทของข้อมูล เพื่อที่ผู้ใช้จะได้สามารถสร้างตัวแปรสำหรับข้อมูลประเภทต่างๆ ได้สอดคล้องกับความต้องการในการใช้งาน บทที่ 3 จะอธิบายถึงตัวดำเนินการที่ใช้สำหรับการคำนวณทางคณิตศาสตร์ เช่น เครื่องหมายการบวก การลบ การคูณ และการหาร เป็นต้น บทที่ 4 จะอธิบายถึงการใช้เรียกงานฟังก์ชันพื้นฐานต่างๆ สำหรับการคำนวณทางคณิตศาสตร์ บทที่ 5 จะกล่าวถึงหลักการเขียนโปรแกรม SCILAB ซึ่งจะทำให้ผู้ใช้สามารถพัฒนาฟังก์ชันใหม่ๆ ขึ้นมาใช้งานร่วมกับโปรแกรม SCILAB ได้อย่างรวดเร็วและมีประสิทธิภาพ บทที่ 6 จะอธิบายการนำเสนอผลการวิจัยหรือผลการทดลองที่ได้มาโดยใช้รูปกราฟทั้งแบบสองมิติและสามมิติ บทที่ 7 จะอธิบายระบบพื้นฐานอินพุตและเอาต์พุตของโปรแกรม SCILAB ซึ่งจะทำให้สามารถพัฒนาโปรแกรมเพื่อใช้ในการอ่านและเขียนข้อมูลระหว่างเครื่องคอมพิวเตอร์กับโปรแกรม SCILAB ได้ บทที่ 8 จะกล่าวถึงขั้นตอนพื้นฐานในการตรวจสอบหาข้อผิดพลาดภายในตัวโปรแกรม และบทที่ 9 ซึ่งเป็นบทสุดท้ายจะยกตัวอย่างการประยุกต์ใช้งานของโปรแกรม SCILAB เพื่อใช้แก้ปัญหาเฉพาะด้านในรูปแบบต่างๆ

ข้าพเจ้าได้นำหนังสือเล่มนี้มาทดลองใช้กับนักศึกษาชั้นปีที่ 1 และ 2 โปรแกรมวิศวกรรมโทรคมนาคม มหาวิทยาลัยราชภัฏนครปฐม พบว่านักศึกษาที่ไม่เคยรู้จักโปรแกรม SCILAB มาก่อนสามารถทำความเข้าใจตามเนื้อหาในแต่ละบทได้โดยง่าย และเมื่อศึกษาครบทั้งเล่มแล้วก็จะช่วยทำให้นักศึกษาสามารถพัฒนาโปรแกรมพื้นฐานขึ้นมาใช้งานได้อย่างรวดเร็วและมีประสิทธิภาพ สำหรับผู้ที่สนใจรายละเอียดการใช้งานและความสามารถของโปรแกรม SCILAB เพิ่มเติม ก็สามารถศึกษาค้นคว้าได้ด้วยตนเองจากเว็บไซต์ http://www.scilab.org

หนังสือเล่มนี้จะไม่สามารถทำให้สำเร็จขึ้นมาได้ ถ้าหากขาดบุคคลต่างๆ ที่คอยให้ความช่วยเหลือและเป็นกำลังใจให้ข้าพเจ้าตลอดมา ดังนั้นข้าพเจ้าขอกราบขอบพระคุณอาจารย์ทุกท่านที่ให้ความรู้และคำปรึกษาตลอดระยะเวลาการศึกษา รวมทั้งทุกๆ คนในครอบครัวของข้าพเจ้า ได้แก่

คุณเกียรติ โควินท์ทวีวัฒน์, คุณพรรณี โควินท์ทวีวัฒน์, คุณรัชนิศ โรจนกิจ, คุณอนุทัศน์ โรจนกิจ, คุณฉัตรชัย โควินท์ทวีวัฒน์, คุณกิตติศักดิ์ โควินท์ทวีวัฒน์, และคุณศิรสุดา โสมนัส นอกจากนี้ ข้าพเจ้าขอขอบคุณมหาวิทยาลัยราชภัฏนครปฐมที่ให้การสนับสนุนและให้ความสะดวกแก่ข้าพเจ้าตลอดระยะเวลาในการเขียนหนังสือเล่มนี้

ท้ายสุดนี้ข้าพเจ้าพยายามอย่างยิ่งในการที่จะทำให้หนังสือเล่มนี้ง่ายต่อการเรียนรู้ เพื่อให้ผู้อ่านสามารถทำความเข้าใจได้ด้วยตนเองอย่างรวดเร็วและมีประสิทธิผล ดังนั้นหากมีข้อบกพร่องประการใด ข้าพเจ้ามีความยินดีและจักขอบพระคุณยิ่ง ถ้าท่านผู้ใช้หนังสือเล่มนี้จะส่งข้อคิดเห็นและคำแนะนำที่เป็นประโยชน์สำหรับการปรับปรุงหนังสือเล่มนี้มาที่อีเมล์ piya@npru.ac.th เพื่อที่ข้าพเจ้าจะได้ดำเนินการปรับปรุงและแก้ไขในการพิมพ์ครั้งต่อไป

ผศ.ดร.ปิยะ โควินท์ทวีวัฒน์
โปรแกรมวิศวกรรมโทรคมนาคม
มหาวิทยาลัยราชภัฏนครปฐม
กรกฎาคม พ.ศ. 2551

สารบัญ

บทที่ 1

ความรู้เบื้องต้นเกี่ยวกับโปรแกรม SCILAB

ในบทนี้จะกล่าวถึงประวัติความเป็นมาของโปรแกรม SCILAB โครงสร้างทั่วไป และคำสั่งพื้นฐานสำหรับการเริ่มต้นใช้งานโปรแกรม SCILAB

1.1 ประวัติความเป็นมาของโปรแกรม SCILAB

SCILAB เป็นโปรแกรมภาษาขั้นสูงที่ถูกพัฒนาขึ้นโดยความร่วมมือกันระหว่างนักวิจัยจากสถาบัน Institut National De Recherche En Informatique Et En Automatique (INRIA) และ École nationale des ponts et chausses (ENPC) ประเทศฝรั่งเศส ตั้งแต่ปี ค.ศ. 1990 โดยมีจุดมุ่งหมายเพื่อใช้ในการคำนวณเชิงตัวเลขและแสดงผลกราฟิกที่ซับซ้อน ดังนั้นโปรแกรม SCILAB จึงเหมาะสำหรับการใช้งานทางด้านวิศวกรรมและวิทยาศาสตร์ นอกจากนี้โปรแกรม SCILAB ยังเป็นโปรแกรมที่ให้ฟรี (ไม่ต้องเสียเงินค่าลิขสิทธิ์ซอฟต์แวร์) และอนุญาตให้ผู้ใช้สามารถนำไปพัฒนาต่อยอดได้ ผู้สนใจสามารถที่จะดาวน์โหลดตัวโปรแกรม SCILAB และข้อมูลที่เกี่ยวข้องต่างๆ ได้จากเว็บไซต์ http://home.npru.ac.th/~t3058/Scilab.htm หรือ http://www.scilab.org

สิ่งที่สำคัญอีกประการหนึ่งของโปรแกรม SCILAB ก็คือความสามารถในการทำงานที่ใกล้เคียงกับโปรแกรม MATLAB[1] ซึ่งเป็นโปรแกรมที่นิยมมากสำหรับผู้ใช้งานทางด้านวิศวกรรมและวิทยาศาสตร์ แต่ค่าลิขสิทธิ์ซอฟต์แวร์ของโปรแกรม MATLAB นั้นมีราคาแพงมาก ดังนั้นในปัจจุบันนี้หลายๆ หน่วยงานทั้งภาคอุตสาหกรรมและภาคการศึกษาทั้งในและนอกประเทศได้เริ่มนำโปรแกรม SCILAB มาช่วยในการทำงานและช่วยในการเรียนการสอน ทั้งนี้เนื่องจากโปรแกรม

[1] ดูรายละเอียดของโปรแกรมภาษา MATLAB ได้จากเว็บไซต์ http://www.mathworks.org

SCILAB เป็นโปรแกรมที่สามารถทำงานได้อย่างมีประสิทธิภาพ และไม่ต้องเสียเงินค่าลิขสิทธิ์ซอฟต์แวร์[2] โดยทั่วไปข้อดีของโปรแกรม SCILAB สามารถสรุปได้ดังนี้

- ง่ายต่อการเรียนรู้และทำความเข้าใจ
- ขั้นตอนการเขียนโปรแกรมไม่ยุ่งยาก
- มีฟังก์ชัน (function) สำหรับการคำนวณทางคณิตศาสตร์จำนวนมากพร้อมใช้งาน
- มีกล่องเครื่องมือ (toolbox) จำนวนมากที่ประกอบด้วยฟังก์ชันต่างๆ ที่จำเป็นสำหรับการแก้ไขปัญหาทางด้านวิศวกรรมและวิทยาศาสตร์ เช่น กล่องเครื่องมือทางด้านการควบคุมทนทาน (robust control), กล่องเครื่องมือทางด้านการประมวลผลสัญญาณ (signal processing), และกล่องเครื่องมือทางด้านการหาค่าเหมาะที่สุด (optimization) เป็นต้น
- สามารถประมวลผลข้อมูลที่อยู่ในรูปเชิงสัญลักษณ์ (symbolic) และข้อมูลที่อยู่ในรูปของเมทริกซ์ (matrix) ได้อย่างรวดเร็วและมีประสิทธิภาพ
- สามารถพัฒนาฟังก์ชันใหม่ๆ ขึ้นมาใช้งานร่วมกับโปรแกรม SCILAB ได้โดยง่าย
- สามารถใช้งานร่วมกับโปรแกรมภาษาฟอร์แทรน (FORTRAN), ภาษาซี (C) และภาษา MATLAB ได้
- สามารถใช้งานร่วมกับโปรแกรม LabVIEW เพื่อทำการเชื่อมต่อกับอุปกรณ์ฮาร์ดแวร์ได้
- สามารถสร้างโปรแกรมสำเร็จรูปสำหรับการจำลองระบบ (system simulation) ได้โดยใช้เครื่องมือของโปรแกรม SCILAB ที่เรียกว่า “`scicos`”
- สามารถนำไปพัฒนาต่อยอดได้ เนื่องจากมีรหัสต้นฉบับ (source code) และคู่มือการใช้งานให้ซึ่งสามารถดาวน์โหลดได้จากเว็บไซต์ http://www.scilab.org

จากที่กล่าวมาข้างต้นนี้จะเห็นได้ว่าโปรแกรม SCILAB สามารถทำงานได้มากมายหลายรูปแบบ อย่างไรก็ตามหนังสือเล่มนี้จะอธิบายเฉพาะพื้นฐานการใช้งานทั่วไปของโปรแกรม SCILAB เพื่อให้ผู้อ่านสามารถนำไปใช้งานได้อย่างถูกต้องและรวดเร็ว พร้อมทั้งเป็นแนวทางให้ผู้อ่านนำโปรแกรมนี้ไปประยุกต์ใช้งานขั้นสูงต่อไป ถ้ามีโอกาสผู้เขียนก็หวังว่าจะได้เขียนหนังสือเกี่ยวกับการใช้งานขั้นสูงของโปรแกรม SCILAB เช่น วิธีการใช้งานร่วมกับโปรแกรมภาษาซี, วิธีการใช้งานร่วมกับ

[2] นอกจากนี้ยังมีโปรแกรมอื่นๆ อีกที่ให้ฟรี (freeware) และทำงานคล้ายกับโปรแกรม SCILAB เช่น โปรแกรม Octave (http://www.octave.org) และโปรแกรม Rlab (http://rlab.sourceforge.net) เป็นต้น แต่จากที่ข้าพเจ้าได้ศึกษาค้นคว้าข้อมูลของโปรแกรมเหล่านี้ดูพบว่า โปรแกรมภาษา SCILAB สามารถทำงานได้อย่างรวดเร็วและมีประสิทธิภาพมากที่สุด

โปรแกรม LabVIEW, วิธีการใช้งาน scicos, และการประยุกต์ใช้โปรแกรม SCILAB ในการแก้ไขปัญหาทางด้านวิศวกรรม เป็นต้น

สำหรับโปรแกรม SCILAB ที่ทางสถาบัน INRIA และ ENPC พัฒนาขึ้นมานี้สามารถนำไปใช้งานได้ในหลายระบบปฏิบัติการ ได้แก่ ระบบปฏิบัติการลีนุกซ์ (GNU/Linux), ระบบปฏิบัติการวินโดวส์ (Windows2000/XP/VISTA), ระบบปฏิบัติการ HP-UX, และระบบปฏิบัติการ Mac OS X เป็นต้น โดยหนังสือเล่มนี้จะอ้างอิงถึงโปรแกรม SCILAB เวอร์ชัน 4.1.1 (เริ่มต้นใช้งานเมื่อวันที่ 9 พฤษภาคม พ.ศ. 2550) ที่ทำงานบนระบบปฏิบัติการวินโดวส์ อย่างไรก็ตามสำหรับผู้ใช้งานโปรแกรม SCILAB เวอร์ชันอื่นหรือที่ทำงานบนระบบปฏิบัติการอื่น ก็ยังสามารถใช้หนังสือเล่มนี้ในการศึกษาและทำความเข้าใจถึงลักษณะการใช้งานของโปรแกรม SCILAB ด้วยตนเองได้ ทั้งนี้เนื่องจากรูปแบบการใช้งานของคำสั่งต่างๆ รวมถึงหลักการเขียนโปรแกรม SCILAB ยังคงมีลักษณะเหมือนกันในทุกเวอร์ชันและทุกระบบปฏิบัติการ[3]

1.2 เริ่มต้นการใช้งานโปรแกรม SCILAB

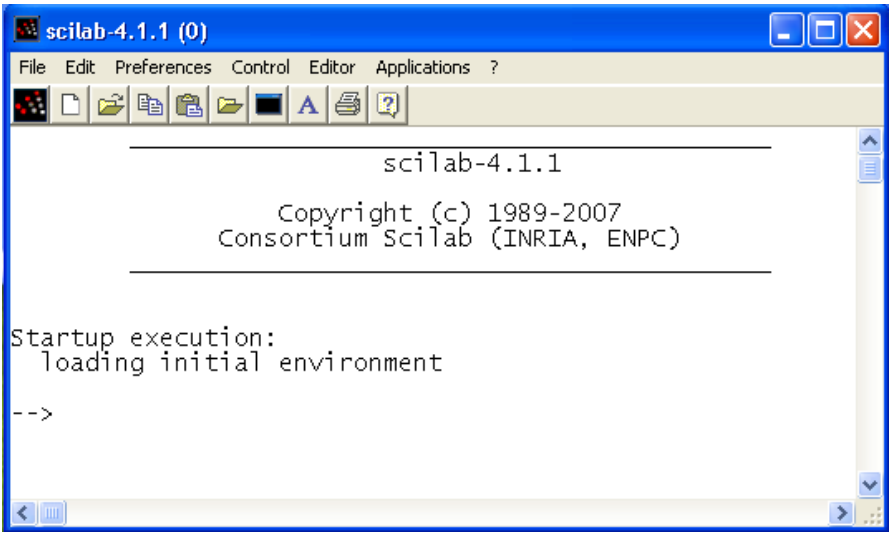
การเรียกใช้งานโปรแกรม SCILAB ให้เรียกจากไฟล์ที่ชื่อว่า “WScilex.exe” ซึ่งอยู่ในสารบบ (directory) “SCIDIR\bin\WScilex.exe” โดยการกดดับเบิลคลิกที่ชื่อไฟล์นี้ เมื่อ SCIDIR คือชื่อสารบบที่ติดตั้งโปรแกรม SCILAB โดยหน้าต่างแรกที่จะถูกแสดงขึ้นมาก็คือ “หน้าต่างคำสั่ง (command window)” ตามรูปที่ 1.1 หน้าต่างนี้จะเป็นส่วนที่ผู้ใช้จะทำการป้อนคำสั่งต่างๆ ลงไปเพื่อทำการคำนวณ และเป็นส่วนที่แสดงผลลัพธ์ออกทางหน้าต่างคำสั่ง โดยที่เครื่องหมาย SCILAB prompt “- - >” เป็นตัวบอกว่าโปรแกรม SCILAB พร้อมที่จะรอรับคำสั่ง ณ จุดนี้

จากรูปที่ 1.1 พบว่าโปรแกรม SCILAB ได้เตรียมแถบเครื่องมือ (toolbar) ที่รวมฟังก์ชันที่ใช้งานบ่อยไว้ที่หน้าต่างคำสั่งในรูปของสัญรูป (icon) ตามที่แสดงในรูปที่ 1.1 ข้างใต้แถบเมนู (menu bar) นั่นคือ  โดยสัญรูปแต่ละอันมีความหมายดังต่อไปนี้

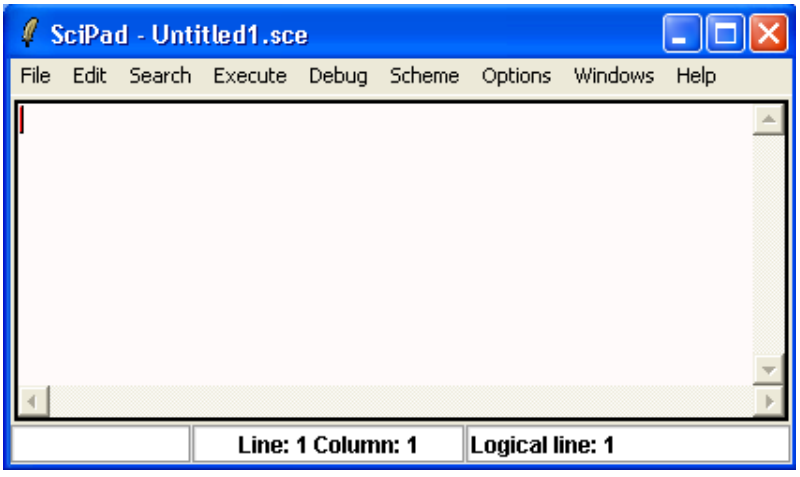
 (New Scilab) สร้างหน้าต่างคำสั่งขึ้นมาใหม่อีกหน้าต่างหนึ่ง

 (Open Scipad) เรียกใช้งานโปรแกรมเอดิเตอร์ (editor) ที่ชื่อว่า “SciPad” ขึ้นมาตามรูปที่ 1.2 เพื่อใช้สำหรับเขียนหรือแก้ไขโปรแกรม

[3] ผู้ใช้สามารถดาวน์โหลดตัวโปรแกรม SCILAB สำหรับระบบปฏิบัติการต่างๆ ได้จาก http://www.scilab.org



รูปที่ 1.1 หน้าต่างคำสั่งของโปรแกรม SCILAB



รูปที่ 1.2 หน้าต่างเอดิเตอร์ที่ชื่อ “SciPad” ของโปรแกรม SCILAB

 (Open file) เรียกไฟล์ที่มีอยู่ในสารบบขึ้นมาใช้งาน

 (Copy) สำเนาข้อมูลในส่วนที่ผู้ใช้เลือกในหน้าต่างคำสั่ง

 (Paste) นำข้อมูลที่สำเนาไว้มาวางในหน้าต่างคำสั่ง

 (Change Directory) เปลี่ยนสารบบที่กำลังทำงาน[4] (working directory)

 (Scilab Output) เรียกหน้าต่างที่ชื่อว่า “Console scilab” ขึ้นมาใช้งาน

 (Choose Font …) เลือกลักษณะของตัวอักษรที่จะใช้ในหน้าต่างคำสั่ง

 (Print) พิมพ์ข้อมูลในหน้าต่างคำสั่งออกทางเครื่องพิมพ์ (printer)

 (Scilab Help) เรียกหน้าต่างให้ความช่วยเหลือที่ชื่อว่า “Scilab Browse Help” ขึ้นมาใช้งานเพื่อค้นหาข้อมูล (มีผลลัพธ์เช่นเดียวกับการป้อนคำสั่ง `help` ในหน้าต่างคำสั่ง)

1.2.1 แถบเมนูในหน้าต่างคำสั่ง

หน้าต่างคำสั่งของโปรแกรม SCILAB จะประกอบไปด้วยเมนูหลักหลายๆ เมนู ซึ่งมีรายละเอียดดังต่อไปนี้

- เมนู **File** ประกอบไปด้วย

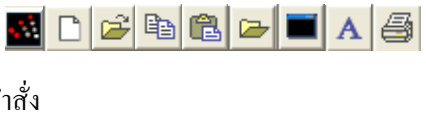
New SCILAB	สร้างหน้าต่างคำสั่งขึ้นมาใหม่อีกหน้าต่างหนึ่ง
Exec …	ประมวลผลชุดคำสั่งจากไฟล์ ซึ่งโดยทั่วไปจะเป็นไฟล์ที่ปิดท้ายด้วย `.sci` หรือ `.sce`
Open …	เรียกไฟล์ที่มีอยู่ในสารบบขึ้นมาใช้งาน
Load …	เรียกข้อมูลที่บันทึกไว้ในไฟล์กลับมาใช้งาน ซึ่งโดยทั่วไปจะเป็นไฟล์ที่ปิดท้ายด้วย `.sav` หรือ `.bin`
Save …	บันทึกข้อมูลของตัวแปรต่างๆ เข้าไปเก็บไว้ในไฟล์ที่กำหนด
Change Directory	เปลี่ยนสารบบที่กำลังทำงาน
Get Current Directory	แสดงสารบบที่กำลังทำงานอยู่ ณ ขณะนั้น
Print Setup …	กำหนดค่าพารามิเตอร์ต่างๆ ของเครื่องพิมพ์
Print …	พิมพ์ข้อมูลออกทางเครื่องพิมพ์
Exit	ปิดหน้าต่างคำสั่ง

[4] สารบบที่กำลังทำงาน คือ สารบบที่โปรแกรม SCILAB กำลังทำงานอยู่ ณ ขณะนั้น กล่าวคือเป็นสารบบที่โปรแกรม SCILAB จะทำการค้นหาไฟล์ต่างๆ หรือบันทึกไฟล์ลงไปเก็บไว้

- เมนู **Edit** ประกอบไปด้วย

Select All	เลือกข้อมูลทั้งหมดในหน้าต่างคำสั่ง
Copy	สำเนาข้อมูลในส่วนที่ผู้ใช้เลือกในหน้าต่างคำสั่ง
Paste	นำข้อมูลที่สำเนาไว้มาวางในหน้าต่างคำสั่ง
Empty Clipboard	ลบข้อมูลที่สำเนาไว้
History	เรียกคำสั่งที่เคยใช้ในหน้าต่างคำสั่งขึ้นมาใช้งานใหม่

- เมนู **Preferences** ประกอบไปด้วย

Language	เลือกภาษาที่จะใช้ในหน้าต่างคำสั่ง (อังกฤษหรือฝรั่งเศส)
Color	เลือกสีของตัวอักษร (text) และสีของพื้นหลัง (background) ที่ต้องการใช้ในหน้าต่างคำสั่ง
Toolbar	เปิดหรือปิดแถบเครื่องมือ  ที่แสดงในหน้าต่างคำสั่ง
Files Association	กำหนดประเภทของไฟล์ที่จะใช้ในหน้าต่างคำสั่ง
Choose Font	เลือกลักษณะของตัวอักษรที่ต้องการจะใช้ในหน้าต่างคำสั่ง
Clear History	ลบคำสั่งต่างๆ ที่เคยใช้ในหน้าต่างคำสั่ง ทำให้ไม่สามารถเรียกใช้งานคำสั่งเหล่านั้นโดยใช้เมนูย่อย History และแป้นลัด (hot key) ได้
Clear Command Window	ลบข้อความทั้งหมดที่ปรากฏในหน้าต่างคำสั่งแล้วให้เครื่องหมาย SCILAB prompt “`-->`” ไปปรากฏอยู่ที่บรรทัดแรกสุดของหน้าต่างคำสั่ง
Console	เรียกหน้าต่าง Console scilab ขึ้นมาใช้งาน

- เมนู **Control** ประกอบไปด้วย

Resume	เรียกใช้คำสั่ง `resume` ของโปรแกรม SCILAB
Abort	เรียกใช้คำสั่ง `abort` ของโปรแกรม SCILAB
Interrupt Ctrl+C	หยุดการประมวลผลชั่วคราว (เทียบเท่ากับการกดปุ่ม Ctrl+C)

- เมนู **Editor**

เป็นการเรียกใช้คำสั่ง `scipad` เพื่อทำการเรียกหน้าต่างเอดิเตอร์ SciPad ขึ้นมา (ตามรูปที่ 1.2) เพื่อใช้ในการเขียนหรือแก้ไขโปรแกรม

- เมนู **Applications** ประกอบไปด้วย

Scicos	เรียกใช้คำสั่ง `scicos` ของโปรแกรม SCILAB
EditGraph	เรียกรูปกราฟที่เก็บไว้มาทำการปรับปรุงหรือแก้ไข
m2sci	เรียกใช้คำสั่ง `mfile2sci` เพื่อทำการแปลง M-file (ไฟล์ฟังก์ชัน[5] ของโปรแกรม MATLAB) ให้เป็นไฟล์ฟังก์ชันของโปรแกรม SCILAB ที่เรียกว่า “SCI-file”
Browser Variables	เรียกใช้คำสั่ง `browsevar` ของโปรแกรม SCILAB เพื่อดูว่า ณ ตอนนี้มีตัวแปรอะไรบ้างที่ได้มีการสร้างขึ้น หรือมีการเรียกใช้งานในหน้าต่างคำสั่งที่กำลังใช้งานอยู่

- เมนู **?** ประกอบไปด้วย

Scilab Help F1	เรียกหน้าต่าง Scilab Browse Help ขึ้นมาใช้งาน
Configure	กำหนดรูปแบบของหน้าต่าง Scilab Browse Help ที่ต้องการจะใช้งานในโปรแกรม SCILAB
Scilab Demos	เรียกดูตัวอย่างการใช้งานชุดคำสั่งต่างๆ ที่โปรแกรม SCILAB ได้จัดเตรียมไว้ให้
Web Links	เปิดใช้งานเว็บไซต์ที่เกี่ยวข้องกับโปรแกรม SCILAB
About	เรียกดูข้อมูลทั่วไปของโปรแกรม SCILAB เช่น เวอร์ชันของโปรแกรม SCILAB ที่กำลังใช้งานอยู่ (เทียบเท่ากับการป้อนคำสั่ง `about` ที่หน้าต่างคำสั่ง)

1.3 คำสั่งพื้นฐานสำหรับการเริ่มต้นใช้งานโปรแกรม SCILAB

โดยทั่วไปการเรียนรู้วิธีการใช้งานโปรแกรมต่างๆ นั้นไม่ยาก ถ้าผู้อ่านศึกษาคู่มือการใช้งานของโปรแกรมที่ต้องการเรียนรู้ พร้อมทั้งทดลองทำตามตัวอย่างทั้งหมดที่ปรากฏอยู่ในคู่มือการใช้งานของโปรแกรมเล่มนั้น ดังนั้นรูปแบบการอธิบายหลักการใช้งานโปรแกรมภาษา SCILAB ของหนังสือเล่มนี้จะเน้นการใช้ตัวอย่างประกอบคำอธิบายเป็นหลัก เพื่อให้ผู้อ่านสามารถเข้าใจลักษณะการใช้งานคำสั่งต่างๆ ได้อย่างรวดเร็วและมีประสิทธิผล

[5] ศึกษารายละเอียดเกี่ยวกับไฟล์ฟังก์ชัน (file function) ได้ในหัวข้อที่ 5.3

ในส่วนนี้จะอธิบายถึงคำสั่งพื้นฐานสำหรับการเริ่มต้นใช้งานโปรแกรม SCILAB เพื่อเป็นแนวทางสำหรับการเรียนรู้ในบทต่อไป เริ่มต้นจะขอแสดงการกำหนดค่าคงที่ให้กับตัวแปร เช่น ถ้าต้องการกำหนดให้ตัวแปร x มีค่าเท่ากับค่า 5 จะต้องป้อนคำสั่งลงในหน้าต่างคำสั่งดังนี้

```
-->x = 5
```

เมื่อกดปุ่ม Enter หน้าต่างคำสั่งจะแสดงผลลัพธ์เป็น

```
x  =
    5.
-->
```

นั่นคือโปรแกรม SCILAB จะแสดงผลลัพธ์ x = 5 ออกมาที่หน้าต่างคำสั่ง พร้อมทั้งแสดงเครื่องหมาย SCILAB prompt “-->” เพื่อรอรับคำสั่งต่อไป

ในกรณีที่ไม่ต้องการให้โปรแกรม SCILAB แสดงผลลัพธ์ออกทางหน้าต่างคำสั่ง ก็ให้ใส่เครื่องหมายเซมิโคลอน (semi-colon) “;” ปิดท้ายคำสั่งนั้น ก่อนกดปุ่ม Enter เช่น ถ้าต้องการกำหนดให้ตัวแปร y มีค่าเท่ากับค่า 3 โดยไม่ต้องแสดงผลลัพธ์ออกทางหน้าต่างคำสั่ง ก็สามารถทำได้โดยป้อนคำสั่งดังนี้

```
-->y = 3;
-->
```

เมื่อกดปุ่ม Enter โปรแกรม SCILAB ก็จะไม่แสดงผลลัพธ์ออกทางหน้าต่างคำสั่ง แต่จะปรากฏเครื่องหมาย SCILAB prompt “-->” แทน

หมายเหตุ ข้อควรระวังในการกำหนดชื่อของตัวแปรมีดังนี้

- โปรแกรม SCILAB จะพิจารณาถึงความแตกต่างของตัวอักษรตัวพิมพ์ใหญ่ (capital letter) กับอักษรตัวพิมพ์เล็ก ซึ่งโปรแกรม SCILAB จะถือว่าตัวแปรที่มีชื่อเหมือนกัน แต่ใช้ตัวอักษรต่างกันจะเป็นตัวแปรคนละตัวกัน ดังนั้นควรหลีกเลี่ยงการกำหนดชื่อตัวแปรที่เหมือนกัน
- ชื่อของตัวแปรจะต้องเริ่มด้วยตัวอักษรและมีความยาวได้ทั้งหมดไม่เกิน 24 ตัวอักษร ทั้งนี้อาจจะใช้ตัวเลขหรือสัญลักษณ์ underline “_” (สัญลักษณ์อื่นห้ามใช้) ช่วยในการกำหนดชื่อของตัวแปรก็ได้

```
break     abort     pause     exit      error     quit
if        then      else      for       while     end
return    resume    function  who       whos      help
sin       cos       tan       atan      exp       log
min       max       median    stdev     det       eye
```

รูปที่ 1.3 ตัวอย่างชื่อที่เป็นคำสำคัญในโปรแกรม SCILAB

- ชื่อของตัวแปรที่กำหนดขึ้นมาจะต้องไม่ซ้ำกับชื่อเฉพาะหรือคำสำคัญ (key word) ที่ใช้ในโปรแกรม SCILAB เช่น ชื่อคำสั่ง และชื่อฟังก์ชัน เป็นต้น ดังตัวอย่างที่แสดงในรูปที่ 1.3
- สำหรับตัวเลขที่มีจำนวนเกินสี่หลัก เช่น ตัวเลขห้าพัน “5,000” เวลากำหนดค่านี้ให้กับตัวแปรจะต้องไม่ใส่เครื่องหมายคอมม่า (comma) คั่นระหว่างเลขห้ากับเลขศูนย์ มิฉะนั้นแล้วโปรแกรม SCILAB จะเข้าใจว่าเป็นเลขสองชุดคือ ค่า 5 กับค่า 0 ตัวอย่างเช่น

```
-->x = [5,000]
 x  =
    5.    0.            //ตัวแปร x เป็นเวกเตอร์ที่มีข้อมูลสองตัวคือ ค่า 5 กับค่า 0

-->x = [5000]
 x  =
    5000.               //ตัวแปร x เป็นสเกลาร์ที่มีค่าเท่ากับ 5000
```

หมายเหตุ ในโปรแกรม SCILAB เครื่องหมาย double slash “//” หรือเครื่องหมายคอมเมนต์ (comment) ที่ปรากฏอยู่ในชุดคำสั่งนี้ จะเป็นเครื่องหมายที่จะบอกให้โปรแกรม SCILAB ไม่ทำการประมวลผลต่อคำสั่งหรือข้อความที่อยู่หลังเครื่องหมายคอมเมนต์นี้ ประโยชน์ของเครื่องหมายคอมเมนต์ก็เพื่อเป็นการเขียนคำอธิบายต่างๆ ภายในตัวโปรแกรมที่พัฒนาขึ้นมาเพื่อเป็นการเตือนความจำของผู้เขียนถึงขั้นตอนการทำงานของตัวโปรแกรม ซึ่งจะทำให้ง่ายต่อการเข้าใจและแก้ไขโปรแกรมในภายหลัง เช่นเดียวกันกับผู้ที่นำโปรแกรมที่พัฒนาขึ้นมานี้ไปใช้งานก็สามารถเข้าใจถึงตัวโปรแกรมได้โดยง่าย

ในกรณีที่ต้องการทราบว่า ณ ขณะนี้มีตัวแปรอะไรบ้างที่ได้มีการสร้างขึ้นหรือที่ได้มีการเรียกใช้งานในหน้าต่างคำสั่งที่ใช้งานอยู่หรือที่เรียกว่า “พื้นที่ใช้งาน (workspace)” สามารถทำได้โดยการใช้คำสั่ง `who` หรือ `whos` โดยที่เมื่อใช้คำสั่ง `who` จะได้ผลลัพธ์เป็น

```
-->who
your variables are...

x          scipad     y          scicos_pal           %scicos_menu
%scicos_short        %scicos_help         %scicos_display_mode
 modelica_libs        scicos_pal_libs      %helps     WSCI       home
 SCIHOME    PWD       TMPDIR     MSDOS     SCI        guilib
 sparselib            xdesslib   percentlib           polylib    intlib
 elemlib    utillib   statslib   alglib    siglib     optlib     autolib
 roblib     soundlib  metalib    armalib   tkscilib   tdcslib    s2flib
 mtlblib    %F        %T         %z        %s         %nan       %inf
 COMPILER   %gtk      %gui       %pvm      %tk        $          %t
%f          %eps      %io        %i        %e
 using      16563 elements  out of    5000000.
           and          58 variables out of       9231

your global variables are...

 LANGUAGE   %helps    demolist   %browsehelp          LCC
%toolboxes           %toolboxes_dir       TMPDIR
 using       1185 elements  out of      11000.
           and           8 variables out of        767
```

ซึ่งจะพบว่ามีตัวแปร x และ y รวมอยู่ในผลลัพธ์ด้วย ทั้งนี้เนื่องจากได้มีการกำหนดค่าของตัวแปร x และ y ไว้ก่อนการเรียกใช้คำสั่ง who นอกจากนี้ยังพบตัวแปรอื่นอีกด้วย ได้แก่ scipad, scicos_pal, %scicos_menu, %help, และ MSDOS เป็นต้น ซึ่งเป็นตัวแปรที่โปรแกรม SCILAB ได้ทำการโหลด (load) เข้าไปเก็บไว้ในหน่วยความจำของเครื่องคอมพิวเตอร์โดยอัตโนมัติตั้งแต่เริ่มเปิดเรียกใช้งานโปรแกรม SCILAB โดยที่ตัวแปรเหล่านี้จะเกี่ยวข้องกับค่าเริ่มต้นของโปรแกรม เช่น ไลบรารี (library), ค่าโดยปริยาย (default) ต่างๆ ของสารบบ, เดโม (demo), และค่าคงที่พิเศษ[6] เป็นต้น

ในขณะที่การใช้คำสั่ง whos จะให้ผลลัพธ์ที่แสดงข้อมูลมากกว่าการใช้คำสั่ง who เช่น บอกว่าตัวแปรแต่ละตัวเป็นข้อมูลประเภท (type) ใด, มีขนาดเท่าใด, และใช้หน่วยความจำกี่ไบต์ (byte) ในการเก็บตัวแปรตัวนั้นๆ ดังแสดงในตัวอย่างต่อไปนี้

[6] ค่าคงที่พิเศษของโปรแกรม SCILAB จะเป็นตัวอักขระที่เริ่มต้นด้วยเครื่องหมายเปอร์เซ็นต์ “%” (สำหรับรายละเอียดเพิ่มเติมของค่าคงที่พิเศษนี้จะมีอธิบายในหัวข้อที่ 2.1)

```
-->whos
Name                    Type          Size          Bytes
whos                    function                    8512
x                       constant      1 by 1        24
scipad                  function                    14312
y                       constant      1 by 1        24
scicos_pal              string        12 by 2       2048
%scicos_menu            list                        2440
%scicos_short           string        12 by 2       408
%scicos_help            sch           ?             62440
%scicos_display_mode    constant      1 by 1        24
modelica_libs           string        1 by 3        528
scicos_pal_libs         string        1 by 11       376
%helps                  *string       28 by 2       16
WSCI                    string        1 by 1        144
home                    string        1 by 1        144
SCIHOME                 string        1 by 1        224
PWD                     string        1 by 1        144
TMPDIR                  string        1 by 1        200
MSDOS                   boolean       1 by 1        16
SCI                     string        1 by 1        112
guilib                  library                     288
sparselib               library                     280
xdesslib                library                     3176
percentlib              library                     13120
polylib                 library                     880
intlib                  library                     1368
elemlib                 library                     2024
utillib                 library                     4688
statslib                library                     1280
alglib                  library                     1336
siglib                  library                     1768
optlib                  library                     608
autolib                 library                     2144
roblib                  library                     1192
soundlib                library                     464
metalib                 library                     3928
armalib                 library                     488
tkscilib                library                     848
[More (y or n ) ?]
```

นอกจากนี้ยังสามารถที่จะตรวจสอบดูได้ว่า ข้อมูลแต่ละตัวที่ใช้เป็นข้อมูลประเภทใดโดยใช้คำสั่ง `typeof` (ดูรายละเอียดการใช้งานคำสั่งนี้ได้ในหัวข้อที่ 2.9)

ถ้าต้องการที่จะทราบเพียงว่ามีตัวแปรอะไรบ้างที่ได้ถูกสร้างขึ้นมาโดยผู้ใช้ ก็สามารถทำได้โดยการใช้คำสั่ง who_user ซึ่งจะให้ผลลัพธ์ดังนี้

```
-->who_user
 User variables are:
 whos      x         scipad    y         modelica_libs
 using 2925 elements out of 4984207
```

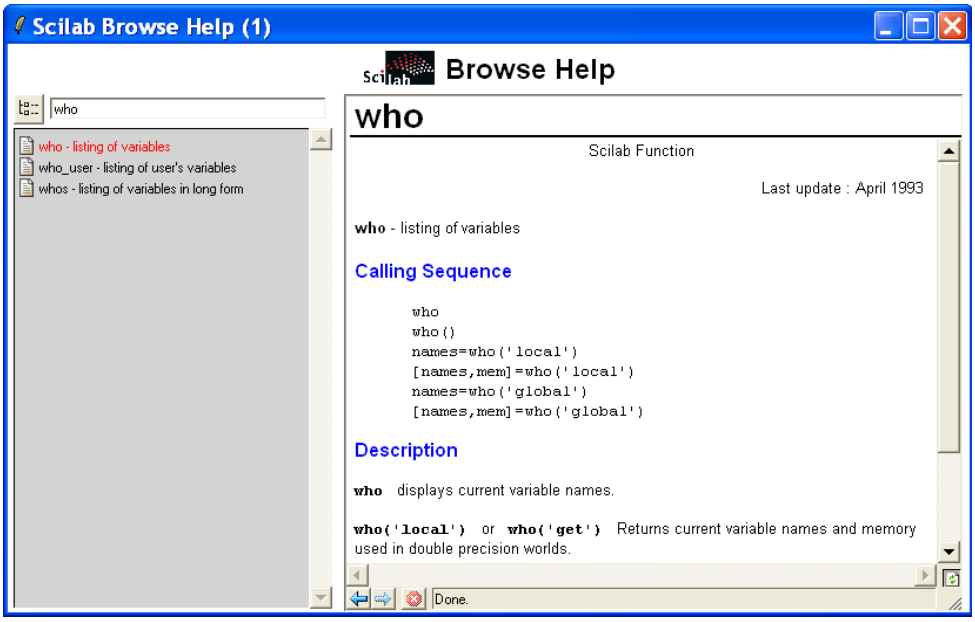
ผลลัพธ์ที่แสดงออกมามีตัวแปร x, y, scipad, และ whos ปรากฏอยู่ด้วยเนื่องจากมีการกำหนดค่าตัวแปร x และ y และมีการเรียกใช้คำสั่ง scipad และ whos ก่อนที่จะทำการเรียกใช้คำสั่ง who_user ส่วนคำสั่ง who ที่ได้มีการเรียกใช้ก่อนหน้านี้โปรแกรม SCILAB ไม่ถือว่าเป็นตัวแปร ในทำนองเดียวกันตัวแปร modelica_libs ถือว่าเป็นค่าเริ่มต้นของโปรแกรม SCILAB

ในกรณีที่ต้องการทราบว่าคำสั่งแต่ละคำสั่งทำหน้าที่อะไร และมีลักษณะการเรียกใช้งานอย่างไร ก็สามารถทำได้โดยการใช้คำสั่ง help เช่น ถ้าต้องการทราบว่าคำสั่ง who มีหน้าที่อะไร ก็ให้ป้อนคำสั่งดังนี้

```
-->help who
```

เมื่อกดปุ่ม Enter ก็จะปรากฏหน้าต่าง Scilab Browse Help ขึ้นมาตามรูปที่ 1.4 ซึ่งจะแสดงข้อความอธิบายรูปแบบการใช้งานของคำสั่ง who จะเห็นได้ว่าคำสั่ง help มีประโยชน์มากสำหรับการเขียนโปรแกรม เพราะว่าคำสั่งแต่ละคำสั่งมีลักษณะการเรียกใช้งานที่ต่างกัน นอกจากนี้คำสั่งหนึ่งคำสั่งอาจจะมีรูปแบบการใช้งานหลายแบบก็ได้ ดังนั้นผู้ใช้จะต้องศึกษาลักษณะการใช้งานคำสั่งต่างๆ ให้เข้าใจเพื่อที่จะได้สามารถนำคำสั่งเหล่านั้นมาใช้งานได้อย่างถูกต้อง และตรงตามความต้องการของงานประยุกต์ (application) ที่กำลังพัฒนา

หากต้องการทราบว่าโปรแกรม SCILAB ได้เตรียมฟังก์ชันอะไรไว้ให้บ้างในแต่ละไลบรารี ก็สามารถทำได้โดยการพิมพ์ชื่อไลบรารีนั้นลงที่หน้าต่างคำสั่งแล้วกด Enter เช่น ถ้าต้องการทราบว่าไลบรารีสำหรับฟังก์ชันพื้นฐาน (elementary function) มีฟังก์ชันอะไรบ้าง ก็ให้ป้อนคำสั่งดังนี้



รูปที่ 1.4 หน้าต่าง Scilab Browse Help ของโปรแกรม SCILAB

```
-->elemlib
elemlib  =

Functions files location :SCI\macros\elem\

 asinh      asinhm     acosh      acosm      acoshm     atanm      asinm
 atanhm     atanh      binomial   convertindex          cellstr    cell2mat
 cosh       cothm      cosm       coshm      cotg       coth       cat
 complex    erf        erfc       erfcx      fix        factor
 factorial             GLoad      isempty    intersect             interpln
 inttrap    integrate             intsplin   isnan      isinf      intc
 intl       ind2sub    isvector   iscellstr             interp1    logm
 log10      lex_sort   log2       modulo     meshgrid   null       nextpow2
 num2cell   pertrans   pmodulo    permute    perms      primes
 squarewave            sqrtm      spzeros    speye      smooth     signm
 setdiff    sinc       sub2ind    sinh       sinhm      sinm       sprand
 toeplitz   tanm       tanhm      tanh       union      unique
 vectorfind
```

หรือถ้าต้องการทราบว่าไลบรารีสำหรับการประมวลผลสัญญาณ (signal processing) มีฟังก์ชันอะไรบ้าง ก็ให้ป้อนคำสั่งดังนี้

```
-->siglib
siglib  =

Functions files location :SCI\macros\signal\

 %asn       analpf     bilt       buttmag    casc       cepstrum   cheb1mag
 cheb2mag   chepol     convol     cspect     czt        dft        detrend
 ell1mag    eqfir      eqiir      faurre     findm      find_freq
 frfit      frmag      fsfirlin   ffilt      fftshift   fft2       group
 hank       hilb       iir        iirgroup   iirlp      iirmod     intdec
 ifft       jmat       %k         kalm       lattn      lev        levin
 lgfft      lindquist             mese       mfft       mrfit      phc
 pspect     remezb     %sn        system     sskf       sincd      srfaur
 srkf       trans      wigner     wiener     wfir       window     yulewalk
 zpch1      zpbutt     zpell      zpch2
```

ผลลัพธ์ที่ได้จะบอกว่าไลบรารีนี้อยู่ที่สารบบใด (เมื่อ SCI คือสารบบที่เก็บโปรแกรม SCILAB ไว้) พร้อมทั้งแสดงฟังก์ชันทั้งหมดที่มีอยู่ในสารบบนี้ ในทางกลับกันถ้าหากต้องการทราบว่าฟังก์ชันนี้อยู่ในไลบรารีใด ก็สามารถทำได้โดยใช้คำสั่ง whereis ตามตัวอย่างต่อไปนี้

```
-->whereis('log10')
 ans  =
 elemlib

-->whereis('convol')
 ans  =
 siglib
```

ในกรณีที่ต้องการจะบันทึกข้อมูลของตัวแปรต่างๆ เข้าไปเก็บไว้ในไฟล์ ก็สามารถทำได้โดยการใช้คำสั่ง save ซึ่งมีรูปแบบการใช้งาน คือ

```
save('filename', [list_of_variables])
```

เมื่ออินพุตอาร์กิวเมนต์ (input argument) หรือตัวแปรส่งผ่าน filename คือชื่อไฟล์ที่ใช้ในการเก็บข้อมูลของตัวแปรซึ่งจะต้องเขียนให้อยู่ภายในเครื่องหมาย single quote “'...'” หรือเครื่องหมาย double quote “"..."” โดยจะต้องระบุเส้นทาง (path) ของไฟล์ที่จะบันทึกข้อมูลให้

ชัดเจนด้วย[7] ถ้าหากไม่ระบุเส้นทางที่จะบันทึกไฟล์ โปรแกรม SCILAB ก็จะทำการบันทึกไฟล์นี้ไว้ในสารบบที่กำลังทำงานอยู่ ณ ขณะนั้นโดยอัตโนมัติ ส่วนอินพุตอาร์กิวเมนต์ที่อยู่ในเครื่องหมายวงเล็บเหลี่ยม (square brackets) "[]" คือพารามิเตอร์ที่เป็นตัวเลือก[8] (option) ของคำสั่งนั้น โดยในที่นี้คือ `list_of_variables` ซึ่งจะเป็นชื่อของตัวแปรที่ต้องการจะบันทึกข้อมูลเข้าไปเก็บไว้ในไฟล์ ถ้าไม่ได้ระบุพารามิเตอร์ `list_of_variables` โปรแกรม SCILAB จะทำการบันทึกข้อมูลของตัวแปรทุกตัวที่ใช้ในหน้าต่างคำสั่งเข้าไปเก็บไว้ในไฟล์ ตัวอย่างการใช้งานคำสั่ง `save` เช่น

```
-->x = 5; y = 3;                  //กำหนดค่าตัวแปร x = 5 และ y = 3
-->save('test.dat',x);            //บันทึกข้อมูลเฉพาะของตัวแปร x ลงไปในไฟล์ชื่อ test.dat
-->clear;                         //ลบตัวแปรทั้งหมดที่เคยสร้างขึ้นมา
-->x
   --error     4
undefined variable : x            //บอกว่าไม่มีข้อมูลของตัวแปร x อยู่ในโปรแกรม SCILAB
```

ชุดคำสั่งนี้จะทำการบันทึกข้อมูลของตัวแปร `x` ลงไปในไฟล์ชื่อ `test.dat` แล้วเก็บไว้ในสารบบที่กำลังทำงาน จากนั้นก็ทำการลบตัวแปรทั้งหมดที่เคยสร้างขึ้นออกจากหน่วยความจำของโปรแกรม SCILAB โดยใช้คำสั่ง `clear`[9] ดังนั้นเมื่อทำการตรวจสอบดูว่า ณ ขณะนี้ ตัวแปร `x` มีค่าเป็นอะไร ก็จะพบว่าไม่มีข้อมูลของตัวแปร `x` ปรากฏอยู่ในโปรแกรม SCILAB

ถ้าต้องการทราบว่า ณ ตอนนี้โปรแกรม SCILAB ทำงานอยู่บนระบบ MSDOS (เช่น ระบบปฏิบัติการวินโดวส์) หรือไม่ ก็สามารถตรวจสอบได้โดยใช้คำสั่ง `MSDOS` ดังนี้

---

[7] เช่นถ้าต้องการบันทึกข้อมูลของตัวแปรทุกตัวลงในไฟล์ชื่อ `test.dat` และเก็บไว้ในสารบบ "C:\SCIDIR" ก็ให้ใช้คำสั่งดังนี้

```
-->save('C:\SCIDIR\test.dat')
```

[8] พารามิเตอร์ที่เป็นตัวเลือกจะมีหรือไม่มีในคำสั่งก็ได้ เนื่องจากโดยทั่วไปจะมีการกำหนดค่าโดยปริยาย (default value) ของพารามิเตอร์ที่เป็นตัวเลือกไว้ให้เรียบร้อยแล้ว

[9] ถ้าต้องการลบข้อมูลของตัวแปรบางตัวก็สามารถทำได้โดยการระบุชื่อตัวแปรนั้นๆ ตามหลังคำสั่ง `clear` เช่น ถ้าต้องการลบข้อมูลเฉพาะของตัวแปร `y` ก็ทำได้โดยการใช้คำสั่ง

```
-->clear y;
```

```
-->MSDOS
MSDOS  =
  T
```

ผลลัพธ์ที่ได้คือ T หรือเป็นจริง (True) ซึ่งหมายความว่าโปรแกรม SCILAB กำลังทำงานอยู่บนระบบ MSDOS จริง ในกรณีที่ต้องการทราบว่าโปรแกรม SCILAB กำลังทำงานอยู่ในสารบบใดก็สามารถตรวจสอบได้โดยใช้คำสั่ง pwd ดังนี้

```
-->pwd
ans  =
 C:\Documents and Settings\User
```

ผลลัพธ์ที่ได้บอกให้ทราบว่าสารบบที่กำลังทำงานคือ "C:\Documents and Settings\User" ซึ่งหมายความว่าเมื่อใช้คำสั่ง save('test.dat') ตามตัวอย่างที่ผ่านมา ไฟล์ test.dat ก็จะถูกนำไปเก็บไว้ในสารบบนี้ ในกรณีที่ต้องการเปลี่ยนสารบบที่กำลังทำงานให้เป็นสารบบอื่นก็สามารถทำได้โดยการใช้คำสั่ง chdir เช่น ถ้าต้องการเปลี่ยนสารบบที่กำลังทำงานให้เป็นสารบบ "C:\" ก็ทำได้โดยใช้คำสั่งต่อไปนี้

```
-->chdir('C:\');              //เปลี่ยนสารบบที่กำลังทำงานให้เป็นสารบบ "C:\"
-->pwd
 ans  =
 C:\
```

ขอให้ย้อนกลับมาที่สารบบที่กำลังทำงานเป็น "C:\Documents and Settings\User" เนื่องจากไฟล์ test.dat อยู่ที่สารบบนี้ ดังนั้นถ้าต้องการเรียกตัวแปรที่เก็บไว้ในไฟล์ test.dat กลับมาใช้งาน ก็สามารถทำได้โดยการใช้คำสั่ง load ซึ่งมีรูปแบบการใช้งาน คือ

```
load('filename')
```

โดยพารามิเตอร์ filename คือชื่อไฟล์ (จะต้องระบุเส้นทางของไฟล์ที่ต้องการเรียกมาใช้งานให้ถูกต้องครบถ้วน) ภายในเครื่องหมาย '...' หรือเครื่องหมาย "..." ตัวอย่างเช่น

```
-->load('test.dat');          //เรียกตัวแปรที่เก็บไว้ในไฟล์ test.dat ในสารบบที่กำลัง

-->x                          //ทำงานกลับมาใช้งาน
 x  =
    5.

-->y
   --error     4
undefined variable : y
```

หลังจากใช้คำสั่ง `load` แล้วจะพบว่า คราวนี้ผู้ใช้สามารถนำค่าของตัวแปร `x` มาใช้งานได้แล้ว แต่ไม่สามารถนำค่าของตัวแปร `y` มาใช้งานได้ เนื่องจากไฟล์ `test.dat` ไม่ได้เก็บข้อมูลของตัวแปร `y` ไว้

หากต้องการลบคำสั่งและข้อความทั้งหมดที่ปรากฏบนหน้าต่างคำสั่ง แล้วให้เครื่องหมาย SCILAB prompt "`-->`" ไปปรากฏอยู่ที่บรรทัดแรกสุดของหน้าต่างคำสั่ง ก็สามารถทำได้โดยใช้คำสั่ง `clc` ซึ่งย่อมาจากคำว่า "Clear Command Window" (แต่ค่าของตัวแปรต่างๆ ที่ผู้ใช้กำหนดไว้ยังสามารถเรียกใช้งานได้) และในกรณีที่ต้องการเลิกใช้งานโปรแกรม SCILAB ก็ให้ทำการปิดหน้าต่างคำสั่งโดยการใช้คำสั่ง `quit` หรือ `exit`

สุดท้ายนี้จะขอกล่าวถึงเครื่องหมายขึ้นบรรทัดใหม่ "`...`" ซึ่งมีลักษณะเป็นจุดที่เรียงต่อกันสามจุด เครื่องหมายนี้มีประโยชน์มากในการเขียนโปรแกรมโดยเฉพาะอย่างยิ่งเมื่อคำสั่งที่ใช้มีความยาวมาก เครื่องหมายนี้เอาไว้ใช้ต่อท้ายคำสั่งเพื่อบอกโปรแกรม SCILAB ว่าคำสั่งในบรรทัดนั้นยังไม่สิ้นสุด ดังนั้นถึงแม้ว่าจะกดปุ่ม Enter หลังเครื่องหมายจุดสามจุดนี้ โปรแกรม SCILAB ก็จะยังไม่นำคำสั่งนั้นไปประมวลผล แต่จะรอรับข้อมูลส่วนที่เหลือที่จะเขียนต่อไปในบรรทัดใหม่จนกระทั่งหมดคำสั่งแล้วกดปุ่ม Enter อีกครั้ง จากนั้นโปรแกรม SCILAB จึงจะเอาข้อความทั้งหมดมารวมกันเป็นประโยคคำสั่งเดียวแล้วค่อยนำเอาไปประมวลผล ให้พิจารณาตัวอย่างต่อไปนี้จะได้เข้าใจถึงลักษณะการทำงานของเครื่องหมายขึ้นบรรทัดใหม่

```
-->x = 5;

-->y = 3;

-->z = x + y          //หาผลบวกของตัวแปร x กับตัวแปร y
 z  =                 //แล้วนำผลลัพธ์ที่ได้ไปบรรจุไว้ในตัวแปรใหม่ที่ชื่อตัวแปร z
    8.
```

```
-->z = x + ...        //หมายถึงยังไม่สิ้นสุดคำสั่ง โปรแกรม SCILAB จะยังไม่นำข้อมูลนี้ไปประมวลผล
-->y                  //เมื่อกดปุ่ม Enter ก็ถือว่าเป็นการสิ้นสุดคำสั่งที่ป้อนจากบรรทัดก่อนหน้านี้
 z   =                //โปรแกรม SCILAB จะนำคำสั่งทั้งหมดคือ z = x + y ไปประมวลผล
    8.
```

จะเห็นได้ว่าผลลัพธ์ที่ได้มีค่าเท่ากัน

1.4 สรุป

ในบทนี้ได้กล่าวถึงประวัติความเป็นมาและการใช้งานคำสั่งพื้นฐานทั่วไปของโปรแกรม SCILAB เพื่อเป็นแนวทางเบื้องต้นในการเรียนรู้ในบทต่อๆ ไป เนื่องจากโปรแกรม SCILAB เป็นโปรแกรมที่ให้**ฟรี**และอนุญาตให้ผู้ใช้สามารถนำไปพัฒนาต่อยอดได้ ดังนั้นโปรแกรม SCILAB จึงถือได้ว่าเป็นทางเลือกใหม่ของผู้ที่ต้องการจะใช้งานโปรแกรมที่มีความสามารถในการคำนวณทางเชิงตัวเลขและแสดงผลกราฟริกที่ซับซ้อน เพื่อใช้ในการแก้ไขปัญหาทางด้านวิศวกรรมและวิทยาศาสตร์ได้อย่างรวดเร็วและมีประสิทธิภาพ เช่นเดียวกันกับโปรแกรม MATLAB

1.5 แบบฝึกหัดท้ายบท

1.1 จงอธิบายคุณสมบัติของโปรแกรม SCILAB

1.2 จงเปรียบเทียบข้อแตกต่างระหว่างโปรแกรม SCILAB และโปรแกรม MATLAB

1.3 จงเปรียบเทียบข้อแตกต่างระหว่างคำสั่ง `who` และคำสั่ง `whos`

1.4 จงอธิบายหน้าที่การทำงานของคำสั่ง `who_users`

1.5 จงทดสอบการใช้งานคำสั่ง `save` และคำสั่ง `load`

1.6 จงอธิบายหน้าที่การทำงานของคำสั่ง `clear`

บทที่ 2

ประเภทของข้อมูล

ในบทนี้จะอธิบายถึงประเภทของข้อมูลประเภทต่างๆ ที่สามารถใช้งานได้ในโปรแกรม SCILAB เช่น ค่าคงที่พิเศษ (special constant), เมทริกซ์ค่าคงที่ (constant matrix), เมทริกซ์ของสายอักขระ (matrix of character strings), พหุนาม (polynomial), และเมทริกซ์บูลีน (Boolean matrix) เพื่อที่ผู้ใช้จะได้สามารถนำข้อมูลแต่ละประเภทมาใช้ในการพัฒนาโปรแกรมได้อย่างถูกต้อง

2.1 ค่าคงที่พิเศษ

ค่าคงที่พิเศษ (special constant) เป็นค่าคงที่ที่ติดตั้งมาพร้อมกับตัวโปรแกรม SCILAB (built-in constant) ไม่สามารถลบทิ้งได้[10] เช่น `%i`, `%pi`, `%e`, `%inf`, `%nan`, `%eps`, และ `ans` เป็นต้น จะเห็นได้ว่าค่าคงที่พิเศษเหล่านี้จะเริ่มต้นด้วยเครื่องหมายเปอร์เซ็นต์ “`%`” แล้วตามด้วยตัวอักขระ ในส่วนนี้จะขอสรุปการใช้งานของค่าคงที่พิเศษต่างๆ ตามตารางที่ 2.1 ดังต่อไปนี้

2.1.1 ค่าคงที่พิเศษ `%i`

ค่าคงที่พิเศษ `%i` คือ ค่าคงที่ที่ใช้ในการแสดงตัวเลขเชิงซ้อน โดยจะมีค่าเท่ากับค่าหน่วยจินตภาพ (imaginary unit) นั่นคือ $\mathtt{i} = \sqrt{-1}$ เช่น ถ้าต้องการกำหนดให้ตัวแปร `a` มีค่าเท่ากับตัวเลข เชิงซ้อน `2+3i` ก็สามารถกำหนดค่าตัวแปร `a` ในโปรแกรม SCILAB ได้ดังนี้

---

[10] สามารถดูรายละเอียดของค่าคงที่พิเศษต่างๆ ที่ใช้ในโปรแกรม SCILAB ว่ามีอะไรบ้าง ได้จากไฟล์เริ่มต้น (start-up file) ซึ่งอยู่ที่สารบบ “SCIDIR/scilab.star”

ตารางที่ 2.1 ค่าคงที่พิเศษในโปรแกรม SCILAB

ค่าคงที่พิเศษ	คำอธิบาย
`%i`	ค่าคงที่ที่ใช้ในการแสดงตัวเลขเชิงซ้อน มีค่าเท่ากับ `%i` $=\sqrt{-1}$
`%pi`	ค่าอัตราส่วนระหว่างความยาวเส้นรอบวงกับเส้นผ่านศูนย์กลางของวงกลม
`%e`	ค่าคงที่ตรีโกณมิติ (trigonometric constant)
`%inf`	ใช้แทนตัวเลขที่มีค่าเป็นอนันต์ (infinity) นั่นคือ `%inf` $=\infty$ เช่น ค่า `1/0`
`%nan`	ใช้แสดงถึงค่าที่ไม่สามารถแสดงให้อยู่ในรูปของตัวเลขได้ (มาจากคำว่า Not-A-Number)
`%eps`	ค่าหน่วยย่อยขนาดเล็กที่สุดที่โปรแกรม SCILAB สามารถรองรับได้ กล่าวคือค่าจำนวนจริงที่ค่าน้อยกว่าค่า `%eps` โปรแกรม SCILAB จะถือว่ามีค่าเป็นค่า `0`
`ans`	ตัวแปรชั่วคราว (temporary variable) ที่เก็บผลลัพธ์ที่ได้จากการคำนวณของคำสั่งซึ่งจะถูกนำมาใช้โดยอัตโนมัติ ในกรณีที่ไม่มีการระบุตัวแปรรับค่าของผลลัพธ์
`%t` หรือ `%T`	ค่าคงที่บูลีน (Boolean constant) ที่แสดงว่าเป็นจริง (True)
`%f` หรือ `%F`	ค่าคงที่บูลีนที่แสดงว่าเป็นเท็จ (False)
`%s` และ `%z`	ตัวแปรที่ใช้ในการกำหนดสมการพหุนาม
`%io`	เป็นตัวอ้างถึงระบบอินพุตและเอาต์พุตของโปรแกรม SCILAB

```
-->%i
 %i  =
    i

-->a = 2 + 3*%i                    //ต้องใส่เครื่องหมายคูณ * หน้าค่าคงที่พิเศษ %i ด้วย
 a  =
    2. + 3.i
```

2.1.2 ค่าคงที่พิเศษ `%pi`

ค่าคงที่พิเศษ `%pi` คือ ค่าอัตราส่วนระหว่างความยาวเส้นรอบวงกับเส้นผ่านศูนย์กลางของวงกลม ซึ่งมีค่าเท่ากับ $\pi = 3.1415927...$ ในโปรแกรม SCILAB ค่า `%pi` จะมีค่าดังนี้

```
-->%pi
 %pi  =
    3.1415927
```

2.1.3 ค่าคงที่พิเศษ %e

ค่าคงที่พิเศษ %e คือ ค่าคงที่ตรีโกณมิติ (trigonometric constant) มีค่าเท่ากับ e = 2.7182818... ในโปรแกรม SCILAB ค่า %e จะมีค่าดังนี้

```
-->%e
 %e  =
    2.7182818
```

2.1.4 ค่าคงที่พิเศษ %inf

ค่าคงที่พิเศษ %inf มาจากคำว่า “infinity” ใช้แทนตัวเลขที่มีค่าเป็นอนันต์ นั่นคือ %inf = ∞ เช่น ผลลัพธ์ที่ได้จากการหารค่าจำนวนจริงด้วยค่าศูนย์ (เช่น 1/0) ในโปรแกรม SCILAB ค่า %inf จะมีค่าดังนี้

```
->%inf
%inf  =
   Inf

-->1/%inf
 ans  =
    0.
```

2.1.5 ค่าคงที่พิเศษ %nan

ค่าคงที่พิเศษ %nan มาจากคำว่า “Not-A-Number” หมายถึงค่าคงที่นั้นไม่สามารถแสดงให้อยู่ในรูปของตัวเลขได้ ในโปรแกรม SCILAB ค่า %nan จะมีค่าดังนี้

```
-->%nan
 %nan  =
   Nan
```

2.1.6 ค่าคงที่พิเศษ %eps

ค่าคงที่พิเศษ %eps มาจากคำว่า “epsilon” คือ ค่าหน่วยย่อยขนาดเล็กที่สุดที่โปรแกรม SCILAB

สามารถรองรับได้ โดยจะมีค่าเท่ากับ e ≈ 2.22*10^{-16} นั่นคือค่าจำนวนจริงที่มีค่าน้อยกว่าค่า %eps โปรแกรม SCILAB จะถือว่าเป็นค่าศูนย์ หรืออาจจะกล่าวได้ว่า %eps เป็นค่าที่มากที่สุดที่ทำให้ x + %eps = x เมื่อ x > 0 ตัวอย่างเช่น

```
-->%eps
 %eps  =
    2.220D-16          //ตัวแปร D-16 หมายถึงให้คูณกับเลขสิบยกกำลังลบสิบหก นั่นคือ 2.22*10^-16
-->2 + %eps
 ans  =
    2.
-->0 + %eps
 ans  =
    2.220D-16
```

ค่าคงที่พิเศษ %eps มีประโยชน์ในการหลีกเลี่ยงปัญหาที่เกิดขึ้นจากการหารด้วยตัวเลขที่มีค่าเป็นศูนย์ เช่น ในกรณีที่ต้องการวาดกราฟของรูปสัญญาณ y = sin(t)/t ซึ่งเป็นสัญญาณที่พบมากในระบบสื่อสารดิจิทัล (digital communication) ตัวอย่างเช่น

```
-->t = -1:0.5:1        //กำหนดให้ตัวแปร t มีค่าตั้งแต่ค่า -1 เพิ่มขึ้นทีละ +0.5 จนถึงค่า +1
 t  =
  - 1.   - 0.5    0.    0.5    1.
-->sin(t)./t
           --error    27
division by zero...          //เนื่องจาก ณ ตำแหน่งที่ t = 0 จะได้ว่า sin(0)/0
-->t(3) = %eps               //กำหนดให้สมาชิกตัวที่สามของตัวแปร t มีค่าเป็น %eps
 t  =
  - 1.   - 0.5    2.220D-16    0.5    1.
-->y = sin(t)./t             //ตามนิยามจะได้ว่า sin(0)/0 = 1
 y  =
    0.8414710    0.9588511    1.    0.9588511    0.8414710
```

สังเกตจะพบว่าตัวเลขที่มีค่ามากๆ หรือน้อยๆ สามารถที่จะเขียนให้อยู่ในรูปของเลขยกกำลังได้โดยใช้พารามิเตอร์ d, e, D, หรือ E เข้าช่วย ดังตัวอย่างต่อไปนี้

```
-->y = [1e-2  1d-3  1D-5  1E-6  10e-4]
 y  =
    0.01    0.001    0.00001    0.000001    0.001

-->z = [1e+2  1d+3  1D+5  1E+6  10e+4]
 z  =
    100.    1000.    100000.    1000000.    100000.
```

2.1.7 ค่าคงที่พิเศษ ans

ค่าคงที่พิเศษ ans คือ ตัวแปรชั่วคราว (temporary variable) ของโปรแกรม SCILAB ที่ใช้ในการเก็บผลลัพธ์ที่ได้จากการคำนวณของคำสั่งแต่ละคำสั่งในหน้าต่างคำสั่ง โดยตัวแปร ans จะถูกนำมาใช้อัตโนมัติในกรณีที่ไม่มีการระบุตัวแปรรับค่าของผลลัพธ์ เช่น

```
-->3+2
 ans  =          //ค่าคงที่พิเศษ ans ถูกนำมาใช้โดยอัตโนมัติ เนื่องจากไม่มีการกำหนดตัวแปรรับค่าผลลัพธ์
    5.
```

2.1.8 ค่าคงที่พิเศษ %t, %T, %f, และ %F

ค่าคงที่พิเศษ %t หรือ %T คือ ค่าคงที่บูลีน (Boolean constant) ที่บอกว่าความสัมพันธ์เป็นจริง (True) หรือมีค่าทางตรรกะเป็นค่า 1 ส่วนค่าคงที่พิเศษ %f หรือ %F คือค่าคงที่บูลีนที่บอกว่าความสัมพันธ์เป็นเท็จ (False) หรือมีค่าทางตรรกะเป็นค่า 0 ตัวอย่างเช่น

```
-->[%t, %T, %f, %F, ~%t, ~%f]
 ans  =
  T T F F F T
```

โดยที่เครื่องหมายทิวด้า (tilda) “~” เป็นตัวดำเนินการตรรกะ (logical operator) ที่แสดงถึงการปฏิเสธ นั่นคือ ~%t = F และ ~%f = T (ดูรายละเอียดในหัวข้อที่ 3.2.2)

2.1.9 ค่าคงที่พิเศษ %s และ %z

ค่าคงที่พิเศษ %s และ %z คือ ตัวแปรที่ใช้ในการกำหนดสมการพหุนาม มีค่าเท่ากับการใช้คำสั่ง s = poly(0, 's') หรือ z = poly(0, 'z') ในโปรแกรม SCILAB (ดูรายละเอียดของการใช้งานคำสั่ง poly ได้ในหัวข้อที่ 2.4.1) เช่น

```
-->%s
 %s  =
    s
-->%z
 %z  =
    z
```

```
-->y = 2*%s^2 - 3*%s + 1                    //มีค่าเท่ากับ y = 1 - 3s + 2s^2
 y  =
                2
    1 - 3s + 2s
```

```
-->x = (3*%z^3 + 2*%z^2)/(2*%z - 1)         //มีค่าเท่ากับ x = (3z^3 + 2z^2)/(2z - 1)
 z  =
      2    3
    2z + 3z
    -------
  - 1 + 2z
```

2.1.10 ค่าคงที่พิเศษ %io

หน้าต่างคำสั่งถือว่าเป็นอุปกรณ์อินพุตและเอาต์พุตมาตรฐาน (standard input and output device) ของโปรแกรม SCILAB ที่ทำหน้าที่ในการรับและส่งข้อมูลกับเครื่องคอมพิวเตอร์ ในโปรแกรม SCILAB จะถือว่าเครื่องมืออินพุตและเอาต์พุตเป็นหน่วยเชิงตรรกะ (logical unit) ซึ่งกำหนดโดยตัวเลขจำนวนเต็ม กล่าวคือหน่วยเชิงตรรกะหมายเลข 5 หมายถึงเมื่อใช้งานหน้าต่างคำสั่งสำหรับเป็นอินพุต และหน่วยเชิงตรรกะหมายเลข 6 หมายถึงเมื่อใช้งานหน้าต่างคำสั่งสำหรับเป็นเอาต์พุต ค่าหน่วยเชิงตรรกะทั้งสองนี้จะถูกเก็บไว้ในค่าคงที่พิเศษ %io ซึ่งมีประโยชน์มากในการทำงานกับระบบอินพุตและเอาต์พุตของโปรแกรม SCILAB ตัวอย่างการใช้งานของคำสั่งนี้ เช่น

```
-->%io
 %io  =

    5.    6.          //เวกเตอร์ขนาด 1×2 ค่า 5 สำหรับอินพุต และค่า 6 สำหรับเอาต์พุต

-->%io(1)             //ค่า 5 สำหรับอินพุต
 ans  =
    5.

-->%io(2)             //ค่า 6 สำหรับเอาต์พุต
 ans  =
    6.
```

นอกจากค่าคงที่พิเศษทั้งหมดที่กล่าวมานี้แล้ว ผู้ใช้ยังสามารถที่จะกำหนดค่าคงที่พิเศษอื่นๆ ขึ้นมาใหม่ได้โดยใช้คำสั่ง `predef` ในโปรแกรม SCILAB สำหรับผู้สนใจลองใช้คำสั่ง `help` เพื่อศึกษารูปแบบการใช้งานของคำสั่ง `predef`

2.2 เมทริกซ์ค่าคงที่

เมทริกซ์ค่าคงที่ (constant matrix) คือ เมทริกซ์ที่มีสมาชิกแต่ละตัวเป็นเลขจำนวนจริง (real number) หรือเลขจำนวนเชิงซ้อน (complex number) โดยทั่วไปแล้วค่าสเกลาร์ (scalar) และเวกเตอร์ (vector) ก็ถือว่าเป็นเมทริกซ์แบบหนึ่ง ดังนั้นในส่วนนี้จะขออธิบายถึงการสร้างตัวแปรสเกลาร์ เวกเตอร์ และเมทริกซ์ ดังนี้

2.2.1 สเกลาร์

สเกลาร์ คือ ค่าที่เป็นได้ทั้งจำนวนจริงหรือจำนวนเชิงซ้อน ค่าสเกลาร์สามารถที่จะถูกกำหนดลงในตัวแปรใดๆ ตามที่ต้องการได้ทันที เช่น

```
-->a = 2 + 3*%i
 a
    2. + 3.i
-->b = 3
 b  =
    3.
```

ตัวแปรสเกลาร์ถือได้ว่าเป็นเมทริกซ์ขนาด 1×1 นั่นคือมีขนาด 1 แถว (row) และ 1 แนวตั้ง (column) ดังนั้นถ้าต้องการตรวจสอบว่าตัวแปรสเกลาร์ a และ b มีขนาดเท่าใด ก็สามารถทำได้โดยการใช้คำสั่ง size ซึ่งมีรูปแบบการใช้งานคือ

```
[m, n] = size(X)
```

โดยที่พารามิเตอร์

- X คือตัวแปรค่าคงที่ใดๆ (เป็นได้ทั้งค่าสเกลาร์ เวกเตอร์ และเมทริกซ์)
- m และ n คือขนาด m×n ของพารามิเตอร์ X เมื่อ m บอกจำนวนแถวของ X และ n บอกจำนวนแนวตั้งของ X

ตัวอย่างการใช้งานคำสั่ง size เช่น

```
-->size(a)
 ans  =
    1.    1.
-->size(b)
 ans  =
    1.    1.
```

ผลลัพธ์ที่ได้หมายความว่าตัวแปร a และ b มีขนาด 1×1 (หรือขนาด 1 แถว และ 1 แนวตั้ง)

2.2.2 เวกเตอร์

เวกเตอร์ คือ เมทริกซ์ขนาดหนึ่งแถวหรือเมทริกซ์ขนาดหนึ่งแนวตั้ง เวกเตอร์แถว (row vector) สามารถสร้างได้โดยการใช้เครื่องหมายคอมม่า (comma) “,” หรือช่องว่าง (space) เป็นตัวแยกสมาชิกแต่ละสมาชิกในเวกเตอร์แถว เช่น

```
-->v = [1, 2, -3]
 v  =
    1.    2.  - 3.
-->v = [1  2  -3]
 v  =
    1.    2.  - 3.
```

ผลลัพธ์ที่ได้คือเวกเตอร์แถว v ที่มีสมาชิกอยู่สามตัว (หรือมีขนาดเท่ากับ 1×3) มีค่าเท่ากับ 1, 2, และ -3 ตามลำดับ เช่นเดียวกันถ้าต้องการทราบว่าเวกเตอร์ v มีขนาดเท่าใดก็สามารถทำได้โดยการใช้คำสั่ง size ดังนี้

```
-->size(v)
ans  =
    1.    3.
```

ซึ่งบอกว่าเวกเตอร์ v มีขนาดเท่ากับ 1×3 (หรือ 1 แถว และ 3 แนวตั้ง) และถ้าต้องการทราบว่าเวกเตอร์ v มีความยาวเท่าใดหรือมีจำนวนสมาชิกทั้งหมดเท่าใด ก็สามารถทำได้โดยการใช้คำสั่ง length ดังนี้

```
 -->length(v)
ans  =
    3.                          //หมายความว่าเวกเตอร์ v มีสมาชิกสามตัว
```

หมายเหตุ การใช้ช่องว่างในการแยกสมาชิกของเวกเตอร์แถวมีข้อควรระวัง ดังแสดงในตัวอย่างต่อไปนี้

```
-->v = [1  +2]
 v  =
    1.    2.
-->v = [1 + 2]
 v  =
     3.
-->v = [1  +2  5 - 4]
 v  =
    1.    2.    1.
```

กล่าวคือถ้าเครื่องหมายบวก (หรือเครื่องหมายลบ) อยู่ติดกับตัวเลข โปรแกรม SCILAB จะถือว่าเครื่องหมายบวก (หรือเครื่องหมายลบ) นั้นเป็นเครื่องหมายแสดงค่าบวก (หรือค่าลบ) ของตัวเลข แต่ถ้าเครื่องหมายบวก (หรือเครื่องหมายลบ) ไม่ได้อยู่ติดกับตัวเลข จะหมายความว่าให้นำตัวเลขที่อยู่ระหว่างเครื่องหมายบวก (หรือเครื่องหมายลบ) มาทำการบวก (หรือการลบ) กัน

ในขณะที่เวกเตอร์แนวตั้ง (column vector) สามารถสร้างได้โดยการหาทรานส์โพส[11] (transpose) ของเวกเตอร์แถวโดยใช้เครื่องหมาย single quote “ ' ” ตามหลังตัวแปรเวกเตอร์แถว หรือสามารถสร้างเวกเตอร์แนวตั้งขึ้นใหม่ได้โดยตรงโดยการใช้เครื่องหมายเซมิโคล่อนเป็นตัวแยกสมาชิกแต่ละสมาชิกในเวกเตอร์แนวตั้ง เช่น

```
-->v = [1 2 3];                    //สร้างเวกเตอร์แถว v
-->v'                              //ใช้ทรานส์โพสกับเวกเตอร์แถวเพื่อให้ได้เป็นเวกเตอร์แนวตั้ง
 ans  =
    1.
    2.
    3.
-->w = [1; 2; -3]                  //สร้างเวกเตอร์แนวตั้งขึ้นมาโดยใช้เครื่องหมายเซมิโคล่อน
 w  =
    1.
    2.
  - 3.
```

นอกจากการกำหนดค่าโดยตรงให้กับเวกเตอร์แล้ว ผู้ใช้ยังสามารถกำหนดค่าของเวกเตอร์ให้มีค่าเพิ่มขึ้นหรือลดลงแบบอัตโนมัติได้ โดยการใช้เครื่องหมายโคล่อน (colon) “ : ” เข้าช่วย ซึ่งมีรูปแบบการใช้งานดังนี้

ชื่อตัวแปร = ค่าเริ่มต้น : ค่าที่เพิ่มขึ้น (หรือค่าที่ลดลง) : ค่าสุดท้าย

ในกรณีที่ไม่มีการกำหนดค่าที่เพิ่มขึ้น (หรือค่าที่ลดลง) โปรแกรม SCILAB จะกำหนดให้เป็นค่าที่เพิ่มขึ้นเท่ากับ +1 โดยอัตโนมัติ (ค่าโดยปริยาย) ตัวอย่างเช่น

```
-->z = 1:2:10          //เริ่มต้นที่ค่า 1 แล้วเพิ่มขึ้นทีละ +2 จนกระทั่งถึงค่าที่มากที่สุดที่ไม่เกิน 10
 z  =
    1.    3.    5.    7.    9.
```

[11] ทรานส์โพสของเวกเตอร์ เป็นการเปลี่ยนเวกเตอร์แนวตั้งให้เป็นเวกเตอร์แถว หรือการเปลี่ยนเวกเตอร์แถวให้เป็นเวกเตอร์แนวตั้ง (ดูรายละเอียดของการทรานส์โพสเพิ่มเติมในหัวข้อที่ 2.2.3.1)

```
-->z = 10:-2.5:0      //เริ่มต้นที่ค่า 10 แล้วลดลงทีละ -2.5 จนกระทั่งถึงค่าที่น้อยที่สุดที่ไม่เกิน 0
 z  =
    10.    7.5    5.    2.5    0.

-->z = 0:5            //เริ่มต้นที่ค่า 0 แล้วค่าเพิ่มขึ้นทีละ +1 จนกระทั่งถึง 5
 z  =
    0.    1.    2.    3.    4.    5.

-->z = 5:0            //ค่าเริ่มต้นที่ 5 ไม่สามารถเพิ่มขึ้นทีละ +1 จนถึง 0 ได้ ดังนั้นผลลัพธ์ที่ได้จึงเป็น
 z  =                 //เมทริกซ์ว่าง (empty matrix) นั่นคือมีจำนวนแถวกับจำนวนแนวตั้งเท่ากับศูนย์
     []
```

2.2.3 เมทริกซ์

เมทริกซ์ขนาด m×n คือ เมทริกซ์ที่มีจำนวนจำนวน m แถว และ n แนวตั้ง เช่น ถ้าต้องการสร้างเมทริกซ์ขนาด 2×3 สามารถสร้างได้ เช่น

```
-->A = [1 2 3; 4 5 6]
 A  =
    1.    2.    3.
    4.    5.    6.
```

โดยที่เครื่องหมายเซมิโคล่อนจะทำหน้าที่ระบุว่าเป็นจุดสิ้นสุดของข้อมูลในแต่ละแถว หากต้องการทราบว่าเมทริกซ์ A มีขนาดเท่าใดก็สามารถทำได้โดยใช้คำสั่ง size ดังนี้

```
-->size(A)
 ans  =
    2.    3.
```

ซึ่งบอกว่าเมทริกซ์ A มีขนาด 2×3 (หรือ 2 แถว และ 3 แนวตั้ง)

นอกจากนี้ผู้ใช้สามารถที่จะอ้างถึงสมาชิกแต่ละตัวในเมทริกซ์ได้โดยตรงตามรูปแบบการใช้งานดังนี้

```
-->b = A(2, 3)
 b  =
    6.
```

คำสั่งนี้เป็นการบอกโปรแกรม SCILAB ให้นำค่าของสมาชิกในแถวที่สองและแนวตั้งที่สามของเมทริกซ์ A ไปบรรจุไว้ในตัวแปร b ในทำนองเดียวกันผู้ใช้ยังสามารถที่จะกำหนดค่าให้แก่สมาชิกแต่ละตัวในเมทริกซ์ได้โดยตรง เช่น

```
-->A(2, 3) = 10
 A  =
    1.    2.    3.
    4.    5.    10.
```

ซึ่งเป็นการกำหนดให้ค่าของสมาชิกในแถวที่สองและแนวตั้งที่สามของเมทริกซ์ A มีค่าเป็นค่า 10 ดังนั้นเมทริกซ์ A จึงมีผลลัพธ์ตามที่แสดงไว้ข้างต้น

ในกรณีที่ต้องการแสดงค่าของสมาชิกทั้งหมดเฉพาะในแถวที่หนึ่งของเมทริกซ์ A ก็ทำได้โดยใช้เครื่องหมายโคล่อน เช่น

```
-->A(1, :)
 ans  =
    1.    2.    3.
```

โดยที่เครื่องหมายโคล่อนจะทำหน้าที่ระบุช่วงทั้งหมดของแนวตั้ง (หรือของแถว ทั้งนี้ขึ้นอยู่กับว่าเครื่องหมายโคล่อนอยู่ ณ ตำแหน่งใดในการอ้างอิงถึง) และเพื่อให้เข้าใจลักษณะการใช้งานของเครื่องหมายโคล่อนมากขึ้น ให้พิจารณาตัวอย่างต่อไปนี้

```
-->A(:, 2)                    //แสดงข้อมูลทุกแถวเฉพาะในแนวตั้งที่สอง
 ans  =
    2.
    5.

-->A(1:2, 2:3)                //แสดงข้อมูลเฉพาะในแถวที่หนึ่งถึงแถวที่สอง
 ans  =                       //และแนวตั้งที่สองถึงแนวตั้งที่สาม
    2.    3.
    5.    10.

-->A(:, [1 3 3])              //แสดงข้อมูลทุกแถวเฉพาะในแนวตั้งที่หนึ่ง แนวตั้งที่สาม
ans  =                        //และแนวตั้งที่สามซ้ำอีกครั้ง
    1.    3.    3.
    4.    10.   10.
```

```
-->A(:, 3) = []                    //ทำการลบข้อมูลทุกแถวเฉพาะในแนวตั้งที่สาม
 A  =
    1.    2.
    4.    5.
```

สังเกตจะพบว่า ณ ตอนนี้เมทริกซ์ A จะมีจำนวนแถวและแนวตั้งเท่ากัน นั่นคือเป็นเมทริกซ์ขนาด 2×2 ในกรณีนี้เมทริกซ์ A จะถูกเรียกว่าเป็นเมทริกซ์จัตุรัส[12] (square matrix) ถ้าต้องการอ้างถึงสมาชิกตัวท้ายสุดในแต่ละแถว (หรือแต่ละแนวตั้ง) ก็สามารถทำได้โดยการให้ใช้เครื่องหมาย “$” ตัวอย่างเช่น

```
-->A(1, $)                         //แสดงข้อมูลในแถวที่หนึ่งและแนวตั้งสุดท้าย
 ans  =
    2.

-->A($, 2)                         //แสดงข้อมูลในแถวสุดท้ายและแนวตั้งที่สอง
 ans  =
    5.
```

นอกจากนี้โปรแกรม SCILAB ยังอนุญาตให้นำเมทริกซ์หลายๆ เมทริกซ์มาประกอบกันเป็นเมทริกซ์ขนาดใหญ่ได้ เช่น

```
-->A = [1 2; 3 4];

-->B = [5 6; 7 8];

-->C = [A B]
 C  =
    1.    2.    5.    6.
    3.    4.    7.    8.

-->D = [A; B]
 D  =
    1.    2.
    3.    4.
    5.    6.
    7.    8.
```

[12] เมทริกซ์จัตุรัส คือ เมทริกซ์ที่มีจำนวนแถวเท่ากับจำนวนแนวตั้ง

2.2.4 การหาทรานส์โพสเมทริกซ์

ทรานส์โพสเมทริกซ์ (matrix transpose) เป็นการเปลี่ยนแนวตั้งให้เป็นแถว และเปลี่ยนแถวให้เป็นแนวตั้ง โปรแกรม SCILAB สามารถทำการทรานส์โพสเมทริกซ์ได้ 2 รูปแบบ คือ

1) ทรานส์โพสแบบสังยุค (conjugate transpose) จะใช้เครื่องหมาย “ ' ” เป็นตัวดำเนินการ โดยทำหน้าที่สร้างทรานส์โพสเมทริกซ์ พร้อมทั้งทำการสังยุคของตัวเลขเชิงซ้อนด้วย
2) ทรานส์โพสแบบธรรมดา (transpose) จะใช้เครื่องหมาย “ .' ” เป็นตัวดำเนินการ โดยจะสร้างเฉพาะทรานส์โพสเมทริกซ์เท่านั้น

ตัวอย่างการใช้งานเช่น

```
-->A = [1+2*%i;  3;  2-%i]                    //สร้างเวกเตอร์แนวตั้ง
 A  =
    1. + 2.i
    3.
    2. -  i

-->A'                                         //ใช้ทรานส์โพสแบบสังยุค
 ans  =
    1. - 2.i    3.    2. + i                  //ทำการสังยุคของตัวเลขเชิงซ้อนด้วย

-->A.'                                        //ใช้ทรานส์โพสแบบธรรมดา
 ans  =
    1. + 2.i    3.    2. - i
```

2.2.5 การหาดีเทอร์มิแนนต์

เมทริกซ์จัตุรัสเท่านั้นที่สามารถจะคำนวณหาค่าดีเทอร์มิแนนต์ (determinant) ได้ ในที่นี้จะยกตัวอย่างการหาดีเทอร์มิแนนต์ของเมทริกซ์แบบง่ายๆ ดังนี้

ถ้ากำหนดให้เมทริกซ์ A มีค่าเท่ากับ

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

เป็นที่ทราบกันว่าดีเทอร์มิแนนต์ของเมทริกซ์ A มีค่าเท่ากับ a*d – c*b ในโปรแกรม SCILAB การหาดีเทอร์มิแนนต์ของเมทริกซ์ A สามารถทำได้โดยการใช้คำสั่ง det(A) ดังนี้

```
-->A = [1 2; 3 4];                    //การหาค่าดีเทอร์มิแนนต์ของเมทริกซ์ขนาด 2×2
-->det(A)                             //det(A) = 1*4 – 3*2 = 4 – 6 = -2
 ans  =
  - 2.
-->B = [1 2 3; 2 -1 4; -3 1 2];
-->det(B)                             //หาค่าดีเทอร์มิแนนต์ของเมทริกซ์ขนาด 3×3
 ans  =
  - 41.
```

2.2.6 การหาอินเวอร์สการคูณของเมทริกซ์

อินเวอร์สการคูณของเมทริกซ์มีประโยชน์มากในการแก้สมการคณิตศาสตร์ เฉพาะเมทริกซ์จัตุรัสเท่านั้นที่สามารถหาอินเวอร์สการคูณของเมทริกซ์ได้ เพราะฉะนั้นถ้าให้ A เป็นเมทริกซ์จัตุรัสใดๆ อินเวอร์สการคูณของเมทริกซ์ A คือ A^{-1} โดยที่ $AA^{-1} = I$ และ I เป็นเมทริกซ์เอกลักษณ์[13] (identity matrix) แบบจัตุรัส ในที่นี้จะยกตัวอย่างการหาอินเวอร์สการคูณของเมทริกซ์ขนาด 2×2 ดังต่อไปนี้

ถ้ากำหนดให้เมทริกซ์ A มีค่าเท่ากับ

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

จากการคำนวณทางคณิตศาสตร์จะได้ว่าอินเวอร์สการคูณของเมทริกซ์ A มีค่าเท่ากับ

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

[13] เมทริกซ์ที่มีสมาชิกที่มีค่าเป็นค่า 1 ปรากฏอยู่ในแนวเส้นทแยงมุมหลัก ส่วนสมาชิกในตำแหน่งอื่นๆ ของเมทริกซ์จะมีค่าเป็นค่า 0 (ดูรายละเอียดของเมทริกซ์เอกลักษณ์ได้ในหัวข้อที่ 4.6.1)

ตัวอย่างเช่น ถ้าให้ $A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$ ดังนั้นอินเวอร์สการคูณของเมทริกซ์ A จะมีค่าเท่ากับ

$$A^{-1}=\frac{1}{-2}\begin{bmatrix}4 & -2\\ -3 & 1\end{bmatrix}=\begin{bmatrix}-2 & 1\\ 1.5 & -0.5\end{bmatrix}$$

จากตัวอย่างที่แสดงข้างต้นจะพบว่า เมทริกซ์ใดที่มีค่าดีเทอร์มิแนนต์ไม่เท่ากับค่า 0 ก็จะสามารถหาค่าอินเวอร์สการคูณของเมทริกซ์ได้ ซึ่งโดยทั่วไปจะเรียกเมทริกซ์ลักษณะนี้ว่า “นอนซิงกูล่าร์เมทริกซ์ (nonsingular matrix)” ส่วนเมทริกซ์ที่ไม่สามารถหาค่าอินเวอร์สของตัวมันเองได้ (กล่าวคือ มีค่าดีเทอร์มิแนนต์เท่ากับค่า 0) จะเรียกว่า “ซิงกูล่าร์เมทริกซ์ (singular matrix)” ในโปรแกรม SCILAB การหาอินเวอร์สการคูณของเมทริกซ์ A สามารถทำได้โดยการใช้คำสั่ง inv(A) เช่น

```
-->A = [1 2; 3 4];          //สร้างเมทริกซ์ A

-->inv(A)                   //หาอินเวอร์สการคูณของเมทริกซ์ A
 ans  =
  - 2.       1.
    1.5    - 0.5

-->A\1                      //เป็นการหาอินเวอร์สการคูณของเมทริกซ์ A อีกรูปแบบหนึ่ง
 ans  =
  - 2.       1.
    1.5    - 0.5

-->1/A                      //เทียบเท่ากับการใช้คำสั่ง A\1
 ans  =
  - 2.       1.
    1.5    - 0.5
```

จะเห็นได้ว่าการหาอินเวอร์สการคูณของเมทริกซ์ นอกจากจะใช้คำสั่ง inv(x) แล้ว ยังสามารถที่จะใช้เครื่องหมายหารในการหาอินเวอร์สการคูณของเมทริกซ์ได้เช่นกัน ส่วนการจะใช้เครื่องหมายหารซ้าย “\” หรือเครื่องหมายหารขวา “/” จะขึ้นอยู่กับตำแหน่งของเมทริกซ์ A ว่าอยู่ด้านไหน

ของเครื่องหมายหาร (ตามที่แสดงในตัวอย่างข้างต้น) การใช้เครื่องหมายหารในการหาอินเวอร์สการคูณของเมทริกซ์มีประโยชน์มากในการเขียนโปรแกรม เพราะช่วยทำให้โปรแกรมมีความกะทัดรัดและประมวลผลได้เร็วขึ้น

การหาอินเวอร์สการคูณของเมทริกซ์บางครั้งผลลัพธ์ที่ได้ของสมาชิกบางตัวที่มีค่าน้อยมาก ดังนั้นโปรแกรม SCILAB จึงได้เตรียมคำสั่ง `clean` เพื่อทำหน้าที่ปรับค่าที่น้อยมากเหล่านั้น (ถ้าค่าสัมบูรณ์น้อยกว่าค่า 10^{-10}) ให้มีค่าเป็นค่า 0 ดังแสดงในตัวอย่างต่อไปนี้

```
-->A = [1 2 1; 2 3 2; 1 4 2];

-->X = A*inv(A)              //จากคุณสมบัติอินเวอร์สการคูณของเมทริกซ์จะได้ว่า AA^-1 = I
 X  =
    1.    2.220D-16    0.
    0.    1.           0.
    0.    0.           1.

-->Y = clean(X)
 Y  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.
```

2.2.7 การสลับตำแหน่งสมาชิกภายในเมทริกซ์

การทำงานกับเมทริกซ์บางครั้งมีความจำเป็นที่จะต้องสลับตำแหน่งสมาชิกภายในเมทริกซ์ (หรือเปลี่ยนรูปร่างของเมทริกซ์) ให้อยู่ในรูปแบบที่ต้องการเพื่อนำไปใช้งานเฉพาะด้าน คำสั่งที่ทำหน้าที่ในลักษณะนี้ได้แก่

2.2.7.1 คำสั่ง `mtlb_fliplr`

เป็นคำสั่งที่ใช้ในการสลับตำแหน่งของสมาชิกในเมทริกซ์จากซ้ายไปขวา สังเกตจะพบว่าคำสั่งนี้จะเริ่มต้นด้วยคำว่า "`mtlb`" ซึ่งเป็นตัวที่บอกว่าคำสั่งนี้มาจากโปรแกรม MATLAB[14] (ในกรณีนี้มาจากคำสั่งในโปรแกรม MATLAB ที่ชื่อว่า `fliplr`) ตัวอย่างเช่น

[14] โปรแกรม SCILAB มีฟังก์ชันในการแปลงโปรแกรมที่เขียนโดยภาษา MATLAB ให้อยู่ในรูปของโปรแกรมที่สามารถนำมาใช้งานในโปรแกรม SCILAB ได้ตามข้อกำหนดของโปรแกรม SCILAB (ดูรายละเอียดเพิ่มเติมที่

```
-->A = [1  2  3  4;  5  6  7  8]
 A  =
    1.    2.    3.    4.
    5.    6.    7.    8.

-->B = mtlb_fliplr(A)              //เมทริกซ์ B ก็คือเมทริกซ์ A ที่สลับตำแหน่งของสมาชิก
 B  =                              //ในแต่ละแถวจากซ้ายไปขวา
    4.    3.    2.    1.
    8.    7.    6.    5.
```

2.2.7.2 คำสั่ง matrix

เป็นคำสั่งที่ใช้ในการเปลี่ยนขนาดของเมทริกซ์ โดยจำนวนสมาชิกของเมทริกซ์ใหม่จะยังคงเท่ากับจำนวนสมาชิกของเมทริกซ์เดิม รูปแบบการใช้งานของคำสั่งนี้ คือ

```
matrix(X, m, n)
```

โดยที่พารามิเตอร์

- X คือ เมทริกซ์ขนาด axb
- m และ n คือ ขนาด mxn (หมายถึง m แถวและ n แนวตั้ง) ของเมทริกซ์ใหม่ โดยที่ผลคูณของ a กับ b จะต้องมีค่าเท่ากับผลคูณของ m กับ n

ก่อนที่จะศึกษาตัวอย่างการใช้งานคำสั่ง matrix จะขอกล่าวถึงลักษณะการจัดเก็บข้อมูลของเมทริกซ์ในโปรแกรม SCILAB ก่อน โดยทั่วไปแล้วข้อมูลในเมทริกซ์จะถูกจัดเก็บให้อยู่ในรูปของเวกเตอร์ โดยที่สมาชิกในเมทริกซ์จะถูกเก็บเรียงจากบนลงล่างและจากซ้ายไปขวา เช่น ถ้าเมทริกซ์ A = [1 2 3; 4 5 6] ข้อมูลที่โปรแกรม SCILAB จัดเก็บจะอยู่ในรูปของเวกเตอร์ชั่วคราว [1 4 2 5 3 6] ดังนั้นเมื่อใช้คำสั่ง matrix โปรแกรม SCILAB ก็จะดึงข้อมูลที่จัดเก็บไว้ในรูปของเวกเตอร์ชั่วคราวทีละตัวจากซ้ายไปขวามาสร้างเป็นเมทริกซ์ใหม่ ให้พิจารณาตัวอย่างต่อไปนี้จะได้เข้าใจถึงการใช้งานคำสั่งนี้มากขึ้น

เมนูหลัก Applications และเมนูย่อย m2sci ในแถบเมนูของหน้าต่างคำสั่ง) นอกจากนี้ในโปรแกรม SCILAB จะมีคำสั่งที่ขึ้นต้นด้วย mtlb หลายคำสั่ง (ศึกษารายละเอียดเพิ่มเติมได้จากการใช้คำสั่ง help mtlb)

```
-->A = [1 2 3; 4 5 6]
 A  =
    1.    2.    3.
    4.    5.    6.          //ข้อมูลที่ถูกจัดเก็บไว้ในเวกเตอร์ชั่วคราว คือ [1 4 2 5 3 6]

-->B = matrix(A, 1, 6)      //สร้างเมทริกซ์ B ขนาดหนึ่งแถวและหกแนวตั้ง จากเมทริกซ์ A
 B  =
    1.    4.    2.    5.    3.    6.

-->C = matrix(A, 3, 3)      //จำนวนสมาชิกของเมทริกซ์ C ไม่เท่ากับของเมทริกซ์ A
                     !--error 60
argument with incompatible dimensions

-->D = matrix(A, 3, 2)      //สร้างเมทริกซ์ D ขนาดสามแถวและสองแนวตั้ง จากเมทริกซ์ A
 D  =
    1.    5.
    4.    3.
    2.    6.
```

คำสั่งสุดท้ายจะเป็นการนำเอาข้อมูลในเวกเตอร์ชั่วคราวมาสร้างเป็นเมทริกซ์ใหม่ขนาด 3×2 โดยข้อมูลที่นำมาใช้ในเมทริกซ์ใหม่จะถูกเรียงจากบนลงล่างและจากซ้ายไปขวา

2.3 เมทริกซ์ของสายอักขระ

สายอักขระ (character string) สามารถสร้างได้โดยการเขียนตัวอักขระภายในเครื่องหมาย single quote '...' หรือเครื่องหมาย double quote "..." เช่น

```
--> x = 'Piya'
 x  =
 Piya
-->y = "_Kovintavewat"
 y  =
 _Kovintavewat
```

ผู้ใช้สามารถที่จะนำสายอักขระสองอันมาต่อกันเป็นอันเดียวได้โดยการใช้เครื่องหมายบวก “+” ดังแสดงในตัวอย่างต่อไปนี้

```
-->x + y
ans  =
 Piya_Kovintavewat
```

การสร้างเมทริกซ์ของสายอักขระ (matrix of character strings) ก็สามารถทำได้โดยวิธีการเดียวกันกับการสร้างเมทริกซ์ทั่วไปตามที่อธิบายไว้ในหัวข้อที่ผ่านมา เช่น

```
-->X = ['a'  "b";  "c"  'd']        //สร้างเมทริกซ์ของสายอักขระ X ขนาด 2×2
 X  =
!a  b  !
!      !
!c  d  !
```

ข้อดีของโปรแกรม SCILAB อีกประการหนึ่งก็คือ ความสามารถในการนำเมทริกซ์ของสายอักขระมาทำการประมวลผลได้ เช่น โปรแกรม SCILAB สามารถทำ symbolic triangularization ของเมทริกซ์ X โดยใช้คำสั่ง `trianfml(X)` นั่นคือการทำให้สมาชิกของเมทริกซ์ X ที่อยู่ใต้เส้นทแยงมุมหลัก (main diagonal) มีค่าเป็นค่าศูนย์ โดยใช้เทคนิคทางคณิตศาสตร์ที่เรียกว่า “การดำเนินการตามแถวขั้นมูลฐาน (elementary row-operation)” ตัวอย่างเช่น

```
-->Y = trianfml(X)              //ผลลัพธ์จากการทำ symbolic triangularization ของเมทริกซ์ X[15]
 Y  =
!c  d          !
!              !
!0  c*b-a*d    !
```

ถ้าต้องการทราบว่าเมทริกซ์ X และ Y จะมีค่าเท่ากับเท่าใด หากมีการกำหนดค่าของตัวแปร a, b, c, และ d ก็สามารถทำได้โดยการใช้คำสั่ง evstr เช่น

```
-->a=1; b=2; c=3; d=4;     //กำหนดค่าของตัวแปรต่างๆ
-->evstr(X)                //ประเมินค่า (evaluate) ของเมทริกซ์ X ตามค่าของตัวแปรที่กำหนดให้
 ans  =
```

[15] สามารถดูรายละเอียดวิธีการทำ symbolic triangularization ของตัวอย่างนี้ได้ในภาคผนวก ก

```
    1.    2.
    3.    4.

-->evstr(Y)              //ประเมินค่าของเมทริกซ์ Y
 ans  =
    3.    4.
    0.    2.
```

หรือสามารถใช้คำสั่ง `eval` แทนคำสั่ง `evstr` ก็ได้ดังนี้

```
-->eval(X)               //ให้ผลลัพธ์เทียบเท่ากับคำสั่ง evstr
 ans  =
    1.    2.
    3.    4.
-->eval(Y)
 ans  =
    3.    4.
    0.    2.
```

2.4 พหุนาม

พหุนาม (polynomial) คือ ผลรวมของจำนวนที่เขียนในรูปการคูณของค่าคงที่กับตัวแปรตั้งแต่ 1 ตัวขึ้นไป โดยที่เลขชี้กำลังของตัวแปรแต่ละตัวมีค่าเป็นศูนย์หรือจำนวนเต็มบวก อย่างไรก็ตามหนังสือเล่มนี้จะพิจารณาเฉพาะกรณีที่พหุนามเป็นฟังก์ชันของตัวแปรเพียงตัวเดียวเท่านั้น นั่นคือโปรแกรม SCILAB จะรองรับพหุนามที่มีรูปแบบดังนี้

$$y = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$$

เมื่อพารามิเตอร์ `x` คือตัวแปรพหุนาม, `a = [a0 a1 a2 ... an]` คือเวกเตอร์ที่มีสมาชิกแต่ละตัวเป็นค่าสัมประสิทธิ์[16] (coefficient) ของพหุนาม, `n` คือดีกรี[17] (degree) ของพหุนาม, และ `y` คือสมการพหุนาม ในโปรแกรม SCILAB ค่าสัมประสิทธิ์และดีกรีของสมการพหุนามนี้สามารถหาได้

[16] ศึกษารายละเอียดการใช้งานคำสั่ง `coeff` ที่ใช้ในการหาค่าสัมประสิทธิ์ได้ในหัวข้อที่ 9.4.3.4

[17] ดีกรีของพหุนาม (degree of polynomial) คือค่าเลขชี้กำลังที่มากที่สุดของพหุนามนั้น

โดยใช้คำสั่ง `coeff(y)` และ `degree(y)` ในส่วนต่อไปนี้จะอธิบายถึงการสร้างสมการพหุนามและเมทริกซ์พหุนาม

2.4.1 สมการพหุนาม

สมการพหุนามสามารถสร้างได้โดยใช้คำสั่ง `poly` ดังนี้

```
y = poly(a, "x", [flag])
```

ซึ่งมีรูปแบบการใช้งานอยู่ 2 แบบ คือ

1) ถ้าพารามิเตอร์ `a` เป็นเวกเตอร์ ผลลัพธ์ที่ได้คือ สมการพหุนาม `y` ที่ถูกกำหนดโดยพารามิเตอร์ `x` และ `flag` เมื่อ `x` คือตัวแปรพหุนาม และ `flag` เป็นตัวเลือกที่มีการเรียกใช้งานดังนี้

   `flag = "coeff"` ให้สร้างสมการพหุนามจากค่าสัมประสิทธิ์ที่กำหนดโดย `a`

   `flag = "roots"` (ค่าโดยปริยาย) หมายถึงให้สร้างสมการพหุนามจากรากหรือคำตอบของสมการพหุนามที่กำหนดโดย `a`

   ตัวอย่างการใช้งานคำสั่ง `poly` ในกรณีนี้ เช่น

```
-->p = poly([1  2], "s")          //สร้างสมการพหุนามจากคำตอบของสมการพหุนาม
 p  =
                2
    2 - 3s + s                    //นั่นคือ s = 1 และ s = 2 เป็นคำตอบของ s^2 - 3s + 2 = 0

-->q = poly([1 2 3], "x", "coeff")   //สร้างสมการพหุนามจากค่าสัมประสิทธิ์
 q  =
                  2
    1 + 2x + 3x

-->c = coeff(q)                   //ค่าสัมประสิทธิ์ของพหุนามจะเรียงจากดีกรีน้อยไปหาดีกรีมาก
 c  =
    1.    2.    3.

-->d = degree(q)
 d  =
    2.                            //ดีกรีของพหุนาม
```

นอกจากนี้ยังสามารสร้างตัวแปรพหุนามได้โดยตรง ด้วยวิธีการต่อไปนี้

```
-->x = poly(0, "s")                    //กำหนดให้ x เป็นตัวแปรพหุนาม
 x  =
    s

-->y = 2 - 3*x + x^2
 y  =
                2
    2 - 3s + s
```

หรือสามารถสร้างพหุนามโดยใช้ค่าคงที่พิเศษ %s หรือ %z ได้ดังนี้

```
-->y = 2 - 3*%s + %s^2
 y  =
                2
    2 - 3s + s

-->y = 2 - 3*%z + %z^2
 y  =
                2
    2 - 3z + z
```

2) ถ้าพารามิเตอร์ a เป็นเมทริกซ์ ผลลัพธ์ที่ได้คือ สมการลักษณะเฉพาะ (characteristic equation) ของเมทริกซ์ a ก่อนที่จะแสดงตัวอย่างการใช้งานคำสั่ง poly สำหรับกรณีนี้จะขออธิบายถึงวิธีการหาสมการลักษณะเฉพาะของเมทริกซ์ A ดังต่อไปนี้

สมการลักษณะเฉพาะของเมทริกซ์ A สามารถหาได้จากการหาค่าดีเทอร์มิแนนต์ของเมทริกซ์ $(A-\lambda I)$ แล้วให้ผลลัพธ์ที่ได้มีค่าเป็นค่า 0 กล่าวคือหาค่า $\det(A-\lambda I)=0$ โดยที่ λ คือค่าคงที่ใดๆ และ I คือเมทริกซ์เอกลักษณ์ที่มีขนาดกับเมทริกซ์ A เช่น ถ้ากำหนดให้เมทริกซ์ A มีค่าเท่ากับ

$$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$

สมการลักษณะเฉพาะของเมทริกซ์ A หาได้จากการแก้สมการต่อไปนี้

$$\det\left(A-\lambda I\right)=0$$

$$\det\left(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}-\lambda\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right)=\det\left(\begin{bmatrix}1-\lambda & 2\\ 3 & 4-\lambda\end{bmatrix}\right)=0$$

$$(1-\lambda)(4-\lambda)-(3)(2)=0$$

$$\lambda^2-5\lambda-2=0$$

ดังนั้นสมการลักษณะเฉพาะของเมทริกซ์ A คือสมการ $\lambda^2-5\lambda-2=0$ ในโปรแกรม SCILAB สามารถหาสมการลักษณะเฉพาะของเมทริกซ์ A ได้โดยใช้คำสั่ง poly ดังนี้

```
-->A = [1  2;   3  4];
-->y = poly(A, "x")
 y  =
              2
  - 2 - 5x + x              //ผลลัพธ์เท่ากันกับสมการ λ^2 − 5λ − 2 เมื่อแทนค่า x = λ
```

ถ้าต้องการหารากหรือคำตอบของสมการลักษณะเฉพาะที่ได้มานี้ ก็สามารถทำได้โดยใช้คำสั่ง roots (ดูรายละเอียดการใช้งานคำสั่ง roots ได้ในหัวข้อที่ 4.1.11) ดังนี้

```
-->x = roots(y)
 x  =
  - 0.3722813
    5.3722813
```

คำตอบของสมการลักษณะเฉพาะนี้โดยทั่วไปจะเรียกกันว่า “ค่าลักษณะเฉพาะ[18] (eigenvalue)” ของเมทริกซ์ A นอกจากนี้โปรแกรม SCILAB สามารถคำนวณหาค่าลักษณะเฉพาะของเมทริกซ์ A ได้โดยตรงจากการใช้คำสั่ง spec ดังนี้

```
-->x = spec(A)              //ได้ผลลัพธ์เท่ากับการใช้คำสั่ง x = roots(y)
 x  =
    5.3722813
  - 0.3722813
```

[18] ค่าลักษณะเฉพาะ (eigenvalue) จะพบมากในงานทางด้านวิศวกรรมต่างๆ

2.4.2 เมทริกซ์พหุนาม

เมทริกซ์พหุนาม (polynomial matrix) คือ เมทริกซ์ที่มีสมาชิกเป็นพหุนาม โปรแกรม SCILAB อนุญาตให้มีการนำตัวแปรพหุนามมาจัดให้อยู่ในรูปของเมทริกซ์ได้เช่นเดียวกันกับค่าสเกลาร์ ซึ่งมีประโยชน์มากสำหรับการใช้งานทางด้านวิศวกรรม โดยวิธีการสร้างเมทริกซ์พหุนามก็จะเหมือนกับการสร้างเมทริกซ์ค่าคงที่ ดังแสดงในตัวอย่างต่อไปนี้

```
-->s = poly(0, 's');

-->A = [1, s;  s, 1+s^2]
 A  =
    1     s
                2
    s     1 + s

-->B = [1/s, 1/(1+s);  1/(1+s), 1/s^2]
 B  =
    1            1
    -          -----
    s          1 + s

      1        1
    -----      -
                2
    1 + s      s
```

นอกจากนี้ยังสามารถนำสมการพหุนามมาดำเนินการทางคณิตศาสตร์ เช่น นำมาบวก ลบ คูณ และหารได้ เช่น (ศึกษารายละเอียดเพิ่มเติมได้ในหัวข้อที่ 3.1.4)

```
-->C = A + B
 C  =
                          2
    1 + s         1 + s + s
    -----         ---------
      s             1 + s

              2         2   4
    1 + s + s     1 + s + s
    ---------     ---------
                        2
      1 + s           s
```

2.5 เมทริกซ์บูลีน

ค่าคงที่พิเศษ %t และ %f สามารถนำมาสร้างเป็นเมทริกซ์บูลีน (Boolean matrix) ได้ดังแสดงในตัวอย่างต่อไปนี้

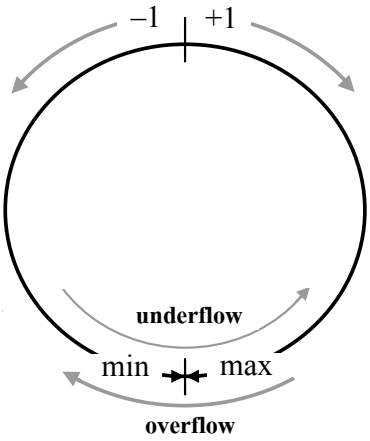
```
-->B = [%t, %T, %f, %F; ~%t, ~%T, ~%f, ~%F]
 B  =
  T T F F
  F F T T

-->[1 2 3] == [2 2 3]              //นำเวกเตอร์สองเวกเตอร์มาเปรียบเทียบกันว่า
 ans  =                            //ที่ตำแหน่งใดบ้างที่มีค่าเท่ากัน
  F T T

-->a = 1:5
 a  =
    1.    2.    3.    4.    5.

-->a > 2                           //ตรวจสอบดูว่าสมาชิกใดของเวกเตอร์ a ที่มีค่ามากกว่าค่า 2
 ans  =
  F F T T T

-->a(ans)                          //แสดงค่าของสมาชิกในเวกเตอร์ a ที่มีค่ามากกว่าค่า 2
 ans  =
    3.    4.    5.
```

จากผลลัพธ์ที่ได้พบว่า มีการนำเอาตัวดำเนินการสัมพันธ์ (relational operator) ได้แก่ เครื่องหมายเท่ากับ “==” และเครื่องหมายมากกว่า “>” เป็นต้น มาใช้ในการสร้างเมทริกซ์บูลีน (ดูรายละเอียดการใช้งานตัวดำเนินการสัมพันธ์ได้ในหัวข้อที่ 3.2.1)

2.6 เมทริกซ์เลขจำนวนเต็ม

โปรแกรม SCILAB รองรับข้อมูลที่เป็นตัวเลขจำนวนเต็ม (integer number) ซึ่งมีอยู่ด้วยกันทั้งหมด 6 แบบ ดังนี้

1) เลขจำนวนเต็มขนาด 8 บิต มีค่าตั้งแต่ -128 ถึง 127 (-2^7 ถึง $2^7 - 1$)



รูปที่ 2.1 รูปแสดงการเปลี่ยนแปลงของข้อมูล เมื่อเกิด overflow และ underflow

2) เลขจำนวนเต็มขนาด 8 บิต แบบไม่คิดเครื่องหมาย มีค่าตั้งแต่ 0 ถึง 255 (0 ถึง $2^8 - 1$)
3) เลขจำนวนเต็มขนาด 16 บิต มีค่าตั้งแต่ -32768 ถึง 32767 (-2^{15} ถึง $2^{15} - 1$)
4) เลขจำนวนเต็มขนาด 16 บิต แบบไม่คิดเครื่องหมาย มีค่าตั้งแต่ 0 ถึง 65535 (0 ถึง $2^{16} - 1$)
5) เลขจำนวนเต็มขนาด 32 บิต มีค่าตั้งแต่ – 2147483648 ถึง 2147483647 (-2^{31} ถึง $2^{31} - 1$)
6) เลขจำนวนเต็มขนาด 32 บิต แบบไม่คิดเครื่องหมาย มีค่าตั้งแต่ 0 ถึง 4294967295 (0 ถึง $2^{32} - 1$)

โดยที่รูปแบบของเลขจำนวนเต็มเหล่านี้สามารถกำหนดได้โดยคำสั่ง int8, uint8, int16, uint16, int32, และ uint32 ตัวอย่างการใช้งานคำสั่งเหล่านี้ เช่น

```
-->int8([-131  -129  -128  0  127  128  130])
 ans  =
  125  127 -128  0  127 -128 -126
-->uint8([-3  -1  0  255  256  258])
 ans  =
  253  255  0  255  0  2
-->int16([-32770  -32768  0 32767  32768  32770])
 ans  =
  32766 -32768  0  32767 -32768 -32766
```

```
-->uint16([-3  -1  0  65535  65536  65538])
 ans  =
  65533  65535  0  65535  0  2

-->int32([-2147483648  0  2147483647  2147483648  2147483649])
 ans  =
 -2147483648  0  2147483647 -2147483648 -2147483648

-->uint32([-3 -1 0 4294967295 4294967296 4294967298])
 ans  =
  4294967293  4294967295  0  4294967295  0  2
```

จะเห็นได้ว่าเมื่อใช้ตัวเลขที่มีค่าเกินกว่าช่วงที่กำหนดไว้ในคำสั่งแต่ละแบบก็จะทำให้เกิดข้อผิดพลาดของข้อมูลซึ่งมีอยู่สองรูปแบบคือ ค่ามากเกินช่วงที่กำหนด (overflow) และค่าน้อยเกินช่วงที่กำหนด (underflow) โดยผลลัพธ์ที่ได้นั้นจะสอดคล้องกับรูปที่ 2.1

จากผลลัพธ์ที่ได้จากการใช้คำสั่งต่างๆ ข้างต้น ถ้าสังเกตจะพบว่าไม่มีจุดทศนิยมตามหลังผลลัพธ์เหล่านั้น ซึ่งทางโปรแกรม SCILAB จะหมายถึงค่าเหล่านั้นเป็นเลขจำนวนเต็ม (ถ้ามีจุดทศนิยมตามหลังตัวเลขจะถือว่าตัวเลขนั้นเป็นเลขจำนวนจริง) ถ้าหากต้องการเปลี่ยนเลขจำนวนเต็มให้เป็นเลขจำนวนจริงที่มีความแม่นยำเป็นสองเท่า (double-precision floating point) ก็สามารถทำได้โดยใช้คำสั่ง `double` เช่น

```
-->uint8([-3  -1  0  255  256  258])
ans  =
  253  255  0  255  0  2

-->double(ans)
 ans  =
    253.    255.    0.    255.    0.    2.    //มีจุดทศนิยมตามหลัง
```

2.7 ลิตซ์

ลิตซ์ (list) คือ กลุ่มของข้อมูลประเภทต่างๆ ที่อยู่รวมกันในตัวแปรตัวเดียว ตัวแปรประเภทลิตซ์จะสามารถอ้างถึงข้อมูลได้หลายประเภทตามที่กำหนด ซึ่งจะมีประโยชน์มากในการเขียนโปรแกรมโดยเฉพาะอย่างยิ่งเมื่อต้องการจัดกลุ่มของข้อมูลประเภทต่างๆ ให้อยู่ภายในชื่อตัวแปรชื่อเดียวกัน คำสั่งที่ใช้ในการกำหนดตัวแปรประเภทลิตซ์มีอยู่สองแบบดังนี้

2.7.1 ลิตซ์แบบธรรมดา

ลิตซ์แบบธรรมดา (ordinary list) สามารถกำหนดให้สมาชิกตัวแรกของตัวแปรประเภทลิตซ์เป็นข้อมูลประเภทไหนก็ได้ การกำหนดลิตซ์แบบนี้จะใช้คำสั่ง `list` ซึ่งมีลักษณะการใช้งาน คือ

```
list(a1, ..., an)
```

โดยที่พารามิเตอร์ `ai` (สำหรับ `i = 1,2,3,...,n`) คือ ข้อมูลประเภทต่างๆ ที่สามารถใช้ได้ในโปรแกรม SCILAB พิจารณาตัวอย่างต่อไปนี้จะได้เข้าใจลักษณะการทำงานของลิตซ์แบบนี้

```
-->L = list(1, 'Piya', [1 2; 3 4])
 L  =

       L(1)          //เป็นข้อมูลสเกลาร์
    1.

       L(2)          //เป็นข้อมูลสายอักขระ
 Piya

       L(3)          //เป็นข้อมูลเมทริกซ์
    1.    2.
    3.    4.
```

จะเห็นได้ว่าตัวแปร `L` ประกอบไปด้วยสมาชิกสามตัว โดยที่สมาชิกแต่ละตัวเป็นข้อมูลคนละประเภท นอกจากนี้ผู้ใช้สามารถที่จะเรียกดูสมาชิกแต่ละตัวได้ดังนี้

```
-->L(2)              //ดูค่าของสมาชิกตัวที่สองในตัวแปร L
 ans  =
 Piya

-->L(3)(1,2)         //ดูค่าของสมาชิกในแถวที่หนึ่งแนวตั้งที่สองของสมาชิกตัวที่สามในตัวแปร L
 ans  =
    2.
```

ถ้าต้องการกำหนดค่าใหม่ให้กับสมาชิกบางตัวในตัวแปร `L` ก็สามารถทำได้ดังนี้

```
-->L(1) = 1 + 2*%s - 3*%s^2;        //กำหนดให้สมาชิกตัวที่หนึ่งเป็นพหุนาม

-->L
 L  =

       L(1)

                2
    1 + 2s - 3s

       L(2)
 Piya

       L(3)
    1.    2.
    3.    4.
```

นอกจากนี้ยังสามารถกำหนดตัวแปร L ให้ซับซ้อนมากขึ้นได้โดยการกำหนดให้สมาชิกบางตัวในตัวแปร L เป็นข้อมูลประเภทลิตซ์ ซึ่งลิตซ์ลักษณะนี้จะถูกเรียกว่า “nested-list” ดังแสดงในตัวอย่างต่อไปนี้

```
-->L(1) = list('W', [1  2  3  4  5]);
-->L(2) = list('Piya', 'Kovintavewat');
-->L
 L  =

       L(1)
        L(1)(1)
 W
        L(1)(2)
    1.    2.    3.    4.    5.
       L(2)
        L(2)(1)
 Piya
        L(2)(2)
 Kovintavewat
       L(3)
    1.    2.
    3.    4.
```

2.7.2 ไทพ์ลิตซ์

ไทพ์ลิตซ์ (typed-list) เป็นลิตซ์ที่สมาชิกตัวแรกจะต้องเป็นสายอักขระหรือเวกเตอร์ของสายอักขระเท่านั้น ส่วนสมาชิกตัวอื่นจะเป็นข้อมูลประเภทใดก็ได้ พิจารณาตัวอย่างการใช้งานของลิตซ์แบบนี้จะได้เข้าใจมากขึ้น

```
-->L = tlist(['Car'; 'Name'; 'Dimensions'], 'Toyota', [1 2; 3 4])
 L  =

       L(1)

!Car          !
!             !
!Name         !
!             !
!Dimensions   !

       L(2)
 Toyota

       L(3)
    1.    2.
    3.    4.
```

ในทำนองเดียวกันผู้ใช้สามารถที่จะเรียกดูสมาชิกแต่ละตัวได้ดังนี้

```
-->L(1)                 //เรียกดูค่าของสมาชิกตัวที่หนึ่งในตัวแปร L
 ans  =
!Car          !
!             !
!Name         !
!             !
!Dimensions   !

-->L(2)                 //เรียกดูค่าของสมาชิกตัวที่สองในตัวแปร L
 ans  =
 Toyota

-->L.Name               //สามารถเรียกดูค่าของสมาชิกตัวที่สองในตัวแปร L ลักษณะนี้ได้เช่นกัน
 ans  =
 Toyota
```

```
-->L(3)(2,2)
 ans  =
    4.

-->L.Dimensions(2,2)          //ได้ผลลัพธ์เช่นเดียวกันกับการใช้คำสั่ง L(3)(2,2)
 ans  =
    4.
```

2.8 อาร์เรย์หลายมิติ

ในการทำงานที่ซับซ้อนบางครั้ง อาจมีความจำเป็นที่จะต้องกำหนดรูปแบบของข้อมูลให้เป็นแบบอาร์เรย์หลายมิติ (N-dimension array) ซึ่งโดยทั่วไปแล้วการสร้างอาร์เรย์หลายมิติมีลักษณะคล้ายกับการสร้างเมทริกซ์ทั่วไป ดังแสดงในตัวอย่างต่อไปนี้

```
-->M(1,3,2) = 8               //กำหนดให้สมาชิกในแถวที่หนึ่งและแนวตั้งที่สาม
 M  =                         //ของเมทริกซ์ชุดที่สองมีค่าเท่ากับ 8
(:,:,1)                       //เมทริกซ์ชุดที่หนึ่ง
    0.    0.    0.
(:,:,2)                       //เมทริกซ์ชุดที่สอง
    0.    0.    8.

-->M(:, :, 1) = [4 5 6]       //กำหนดให้เมทริกซ์ชุดที่หนึ่งมีค่าเท่ากับ [4 5 6]
 M  =
(:,:,1)
    4.    5.    6.
(:,:,2)
    0.    0.    8.

-->M(1, 3, :)                 //แสดงข้อมูลในแถวที่หนึ่งและแนวตั้งที่สามของเมทริกซ์ทั้งสองชุด
 ans  =
(:,:,1)
    6.
(:,:,2)
    8.
```

```
-->size(M)
 ans  =
    1.    3.    2.
```

นอกจากนี้ยังสามารถสร้างเมทริกซ์หลายมิติได้โดยใช้คำสั่ง hypermat ซึ่งมีรูปแบบการเรียกใช้งาน คือ

```
M = hypermat(dims, [entries])
```

โดยที่พารามิเตอร์

- dims คือ เวกเตอร์แสดงขนาดของเมทริกซ์หลายมิติ
- entries คือ เวกเตอร์ที่เก็บค่าของสมาชิกแต่ละตัวในเมทริกซ์หลายมิติ (ค่าโดยปริยายคือ 0) นอกจากนี้ผลคูณของขนาดทั้งหมดของเมทริกซ์หลายมิติจะต้องเท่ากับจำนวนของข้อมูลในเวกเตอร์ entries

ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->M = hypermat([2 3 2], 1:12)     //ผลคูณของขนาด 2*3*2 = 12
 M  =
(:,:,1)
    1.    3.    5.
    2.    4.    6.
(:,:,2)
    7.    9.    11.
    8.    10.   12.
```

ในโปรแกรม SCILAB เมทริกซ์หลายมิติจะถูกเข้ารหัสเป็นข้อมูลประเภท mlists โดยที่มีตัวอ้างอิงแบบอัตโนมัติอยู่สองตัว คือ dims ซึ่งเป็นพารามิเตอร์ที่ใช้แสดงขนาดของเมทริกซ์หลายมิติ และ entries ซึ่งเป็นพารามิเตอร์ที่ใช้แสดงข้อมูลทั้งหมดที่อยู่ในเมทริกซ์หลายมิติ ตัวอย่างเช่น

```
-->M = hypermat([2 3 2], 1:12);
-->M.dims                          //แสดงขนาด (dimension) ของเมทริกซ์ M
```

```
 ans  =
  2   3   2

-->M.entries                                  //แสดงข้อมูลทั้งหมดภายในเมทริกซ์ M
 ans  =
    1.
    2.
    3.
    4.
    5.
    6.
    7.
    8.
    9.
    10.
    11.
    12.
```

2.9 การตรวจสอบประเภทของข้อมูล

จากที่กล่าวมาในบทนี้จะพบว่าโปรแกรม SCILAB สามารถทำงานกับข้อมูลได้หลายประเภท เช่น ค่าคงที่, เมทริกซ์พหุนาม, เมทริกซ์บูลีน, และลิตซ์ เป็นต้น โดยทั่วไปผู้ใช้สามารถที่จะตรวจสอบดูได้ว่าตัวแปรแต่ละตัวเป็นข้อมูลประเภทใด โดยใช้คำสั่ง `typeof` ซึ่งมีลักษณะการใช้งานดังนี้

```
t = typeof(object)
```

โดยที่พารามิเตอร์ `object` คือ ตัวแปรที่ต้องการจะตรวจสอบว่าเป็นประเภทใด ส่วนเอาต์พุตอาร์กิวเมนต์ (output argument) `t` จะแสดงประเภทของตัวแปร ซึ่งค่าที่เป็นไปได้มีดังนี้

1) `"constant"` ถ้า `object` เป็นเมทริกซ์ค่าคงที่ (จำนวนจริงหรือจำนวนเชิงซ้อน)
2) `"polynomial"` ถ้า `object` เป็นเมทริกซ์พหุนาม
3) `"function"` ถ้า `object` เป็นฟังก์ชัน
4) `"handle"` ถ้า `object` เป็นตัวควบคุมระบบอินพุตและเอาต์พุต
5) `"string"` ถ้า `object` เป็นเมทริกซ์สายอักขระ

6) "boolean" ถ้า object เป็นเมทริกซ์บูลีน

7) "list" ถ้า object เป็นข้อมูลประเภทลิสต์ (ถ้าหาก object เป็นข้อมูลแบบ tlist ค่าพารามิเตอร์ t ที่ได้จะเป็นตัวอักขระตัวแรกของสมาชิกตัวแรกในลิสต์นั้น)

8) "hypermat" ถ้า object เป็นข้อมูลประเภท mlist

9) "rational" ถ้า object เป็นเมทริกซ์ตรรกยะ (rational matrix) โดยทั่วไปมักจะใช้แสดงถึงฟังก์ชันถ่ายโอน (transfer function) ซึ่งใช้งานกันมากในทางวิศวกรรมระบบควบคุม

10) "state-space" ถ้า object เป็นข้อมูลประเภทปริภูมิสถานะ[19]

11) "sparse" ถ้า object เป็นเมทริกซ์มากเลขศูนย์[20]

12) "boolean sparse" ถ้า object เป็นเมทริกซ์ Boolean sparse

ตัวอย่างการใช้งานคำสั่ง typeof มีดังนี้

```
-->typeof(3)
 ans  =
 constant

-->typeof(poly(0, 'x'))
 ans  =
 polynomial

-->deff('y=f(x)', 'y=2*x')        //นิยามฟังก์ชันขึ้นมาใช้งาน (ดูรายละเอียดในหัวข้อที่ 5.3.4)

-->typeof(f)
 ans  =
 function

-->typeof('Piya Kovintavewat')
 ans  =
 string

-->typeof(1/poly(0, 'x'))
```

[19] ปริภูมิสถานะ (state-space) คือ เทคนิคการคำนวณทางคณิตศาสตร์แบบหนึ่งซึ่งใช้มากในทางวิศวกรรม

[20] เมทริกซ์มากเลขศูนย์ (sparse matrix) คือ เมทริกซ์ที่มีสมาชิกที่มีค่าเท่ากับค่า 0 อยู่เป็นจำนวนมาก

```
 ans  =
 rational

-->typeof(%t)
 ans  =
 boolean

-->m = sprand(50, 50, 0.001);     //สร้างเมทริกซ์มากเลขศูนย์แบบสุ่มขนาด 50×50

-->typeof(m)
 ans  =
 sparse

-->typeof(m>m)
 ans  =
 boolean sparse

-->L = tlist(['V', 'A', 'B'], 10, 'Piya');

-->typeof(L)
 ans  =
 V                                  //แสดงอักขระตัวแรกของสมาชิกตัวแรกใน tlist

-->M = hypermat([2 3 2], 1:12);

-->typeof(M)
 ans  =
 hypermat
```

นอกจากนี้ยังมีคำสั่งที่ทำหน้าที่คล้ายกับคำสั่ง `typeof` นั่นคือ คำสั่ง `type` ซึ่งมีรูปแบบการใช้งานดังนี้

```
n = type(object)
```

โดยผลลัพธ์ที่ได้จะเป็นเลขจำนวนเต็ม `n` ที่แสดงถึงประเภทของข้อมูล ตามที่โปรแกรม SCILAB กำหนดไว้ดังนี้

- `n = 1` ถ้า `object` เป็นเมทริกซ์ค่าคงที่ (จำนวนจริงหรือจำนวนเชิงซ้อน)
- `n = 2` ถ้า `object` เป็นเมทริกซ์พหุนาม (polynomial matrix)
- `n = 4` ถ้า `object` เป็นเมทริกซ์บูลีน (Boolean matrix)

- `n = 5` ถ้า `object` เป็นเมทริกซ์มากเลขศูนย์ (sparse matrix)
- `n = 6` ถ้า `object` เป็นเมทริกซ์ Boolean sparse
- `n = 8` ถ้า `object` เป็นเมทริกซ์ของเลขจำนวนเต็มที่ใช้พื้นที่ของหน่วยความจำในการเก็บข้อมูลเท่ากับ 1, 2, หรือ 4 ไบต์
- `n = 9` ถ้า `object` เป็นเมทริกซ์ของตัวควบคุมกราฟริก (graphic handles)
- `n = 10` ถ้า `object` เป็นเมทริกซ์สายอักขระ
- `n = 11` ถ้า `object` เป็นฟังก์ชันที่ยังไม่ได้แปลโปรแกรม (un-compiled function)
- `n = 13` ถ้า `object` เป็นฟังก์ชันที่ถูกแปลโปรแกรมแล้ว (compiled function)
- `n = 14` ถ้า `object` เป็นฟังก์ชันไลบรารี (library function)
- `n = 15` ถ้า `object` เป็นข้อมูลประเภท `list`
- `n = 16` ถ้า `object` เป็นข้อมูลประเภท `tlist`
- `n = 17` ถ้า `object` เป็นข้อมูลประเภท `mlist`
- `n = 128` ถ้า `object` เป็นตัวชี้หรือพอยต์เตอร์ (pointer)

ตัวอย่างการใช้งานคำสั่ง `type` เช่น

```
-->type(3)
 ans  =
    1.

-->type(poly(0,'x'))
 ans  =
    2.

-->type([%t %f])
 ans  =
    4.

-->m = sprand(50, 50, 0.001);
-->type(m)
 ans  =
    5.
-->type(m>m)
 ans  =
    6.
```

```
-->type('Piya Kovintavewat')
 ans  =
    10.

-->deff('y=f(x)', 'y=2*x');
-->type(f)
 ans  =
    13.

-->type(elemlib)
 ans  =
    14.

-->L = tlist(['V', 'A', 'B'], 10, 'Piya');
-->type(L)
 ans  =
    16.

-->type(hypermat([2 3 2], 1:12))
 ans  =
    17.
```

2.10 ตัวอย่างการคำนวณ

ในส่วนนี้จะยกตัวอย่างการแก้ไขปัญหาโจทย์ทางคณิตศาสตร์โดยใช้โปรแกรม SCILAB ดังต่อไปนี้

ตัวอย่างที่ 1 จงหาค่าของฟังก์ชัน $f(x) = 2x - 3$ เมื่อ x มีค่าเท่ากับ –5, 0, และ 5

วิธีทำ คำตอบของโจทย์ข้อนี้หาได้จากการใช้ชุดคำสั่งดังนี้

```
-->x = [-5 0 5];
-->fx = 2*x - 3
 fx  =
  - 13.  - 3.    7.
```

นั่นคือ f(x) มีค่าเท่ากับ -13, -3, และ 7 เมื่อ x มีค่าเท่ากับ –5, 0, และ 5 ตามลำดับ

ตัวอย่างที่ 2 จงพิสูจน์คุณสมบัติของเมทริกซ์ว่า (A + B)C = AC + BC เมื่อ

$$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},\ B=\begin{bmatrix}5 & 6\\ 8 & 7\end{bmatrix},\ \text{และ}\ C=\begin{bmatrix}-1\\ 2\end{bmatrix}$$

วิธีทำ ผู้ใช้สามารถพิสูจน์คุณสมบัติของเมทริกซ์นี้ได้จากการใช้ชุดคำสั่งดังนี้

```
-->A = [1 2; 3 4];
-->B = [5 6; 7 8];
-->C = [-1; 2];
-->y = (A+B)*C
 y  =
    10.
    14.
-->z = A*C + B*C
 z  =
    10.
    14.
```

ซึ่งจะได้ผลลัพธ์เท่ากัน ดังนั้นจึงสรุปได้ว่าเมทริกซ์ (A + B)C = AC + BC

ตัวอย่างที่ 3 กำหนดให้เมทริกซ์

$$A=\begin{bmatrix}1 & 2 & 3\\ 2 & 1 & 2\\ 3 & 2 & 1\end{bmatrix}$$

จงหาค่าดีเทอร์มิแนนต์, อินเวอร์สการคูณ, สมการลักษณะเฉพาะ, และคำตอบของสมการลักษณะเฉพาะที่ได้

วิธีทำ คำตอบของโจทย์ข้อนี้สามารถหาได้จากการใช้ชุดคำสั่งดังนี้

```
-->A = [1  2  3;   2  1  2;   3  2  1];
```

```
-->d = det(A)                          //หาดีเทอร์มิแนนต์
 d  =
    8.

-->Ai = inv(A)                         //หาอินเวอร์สการคูณของเมทริกซ์ A
 Ai  =
  - 0.375    0.5    0.125
    0.5    - 1.     0.5
    0.125    0.5  - 0.375

-->I = clean(A*Ai)                     //ตรวจคำตอบอินเวอร์สการคูณของเมทริกซ์
 I  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.

-->y = poly(A, 'x')                    //หาสมการลักษณะเฉพาะ
 y  =
                2   3
  - 8 - 14x - 3x + x

-->z = roots(y)                        //หาคำตอบของสมการลักษณะเฉพาะ
 z  =
  - 0.7015621
  - 2.
    5.7015621

-->z2 = spec(A)                        //หาคำตอบของสมการลักษณะเฉพาะโดยใช้คำสั่ง spec
 z2  =
  - 2.
  - 0.7015621
    5.7015621
```

ตัวอย่างที่ 4 จงหาคำตอบของสมการพหุนาม $x^4 - 10x^3 + 35x^2 - 50x + 24 = 0$

วิธีทำ คำตอบของสมการพหุนามนี้หาได้จากการใช้ชุดคำสั่งดังนี้

```
-->x = poly(0, 'x');
```

```
-->y = x^4 - 10*x^3 + 35*x^2 - 50*x + 24;        // "^ " คือเครื่องหมายเลขชี้กำลัง

-->z = roots(y)
 z  =
    1.
    2.
    3.
    4.
```

ดังนั้นคำตอบของสมการพหุนามนี้คือ x = 1, 2, 3, และ 4

ตัวอย่างที่ 5 จงหาสร้างสมการพหุนามให้อยู่ในรูปของตัวแปร x โดยให้คำตอบของสมการนี้มีค่าเท่ากับ $x = \{2, -2, 1 + i, 1 - i\}$ เมื่อ $i = \sqrt{-1}$

วิธีทำ สมการพหุนามที่มีคำตอบตามที่โจทย์กำหนดสามารถหาได้ดังนี้

```
-->x = [2, -2, 1+%i, 1-%i]
 x  =
    2.  - 2.    1. + i      1. - i

-->y = poly(x, 'x')
 y  =
                2     3    4
  - 8 + 8x - 2x - 2x + x
```

นั่นคือ $x = \{2, -2, 1 + i, 1 - i\}$ คือ คำตอบของสมการ $x^4 - 2x^3 - 2x^2 + 8x - 8 = 0$

2.11 สรุป

ในบทนี้ได้อธิบายวิธีการสร้างข้อมูลประเภทต่างๆ เช่น ค่าคงที่พิเศษ, เมทริกซ์ค่าคงที่, เมทริกซ์สายอักขระ, พหุนาม, เมทริกซ์บูลีน, และลิตซ์ เป็นต้น พร้อมทั้งแสดงตัวอย่างการใช้โปรแกรม SCILAB ในการแก้ไขปัญหาโจทย์ทางคณิตศาสตร์ ซึ่งเมื่อเข้าใจถึงลักษณะของข้อมูลแต่ละประเภทแล้วก็จะทำให้สามารถสร้างตัวแปรสำหรับจัดเก็บข้อมูลแต่ละประเภทได้อย่างถูกต้องและตรงกับความต้องการในการใช้งาน

2.12 แบบฝึกหัดท้ายบท

2.1 จงคำนวณหาค่าของฟังก์ชัน $f(x) = -5x + 3$ เมื่อ x มีค่าเท่ากับ –10, –5, 0, 5, และ 10

2.2 กำหนดให้

$$A = \begin{bmatrix} 5 & 8 & 7 \\ 4 & 2 & 6 \\ 9 & 8 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 2 & -3 \\ 2 & -4 & 5 \\ 2 & 3 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 3 & 1 & 2 \\ -1 & 2 & 1 \\ -2 & -1 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \quad \text{และ} \quad E = \begin{bmatrix} -2 \\ 0 \\ 2 \end{bmatrix}$$

จงหาค่าต่อไปนี้

2.2.1) A + B – C

2.2.2) AB + BC

2.2.3) (A + B)\C

2.2.4) (A + B)/C

2.2.5) (A + B)C

2.2.6) CA + CB

2.2.7) DE

2.2.8) AEDC

2.3 กำหนดให้

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad B = \begin{bmatrix} -1 & 2 & 1 \\ 2 & 4 & 6 \\ 5 & 3 & 1 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 1 & 2 \\ -1 & 2 & 1 \\ -2 & -1 & 1 \end{bmatrix}$$

จงพิสูจน์ว่า

2.3.1) (AB)C = A(BC)

2.3.2) A(B + C) = AB + AC

2.3.3) $(A^T)^T = A$ โดยที่ $(.)^T$ คือ เครื่องหมายทรานส์โพสของเมทริกซ์

2.3.4) $(kA)^T = kA^T$ โดยที่ k คือ ค่าคงที่ใดๆ

2.3.5) $(A + B)^T = A^T + B^T$

2.3.6) $(AB)^T = B^TA^T$

2.3.7) $AB \neq BA$

2.4 กำหนดให้

$$A = \begin{bmatrix} 5 & 8 & 7 \\ 4 & 2 & 6 \\ 9 & 8 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} -1 & 2 & -3 \\ 2 & -4 & 5 \\ 2 & 3 & 1 \end{bmatrix}$$

จงพิสูจน์ว่า

2.4.1) $\det(A) = \det(A^T)$ โดยที่ det(.) คือ เครื่องหมายดีเทอร์มิแนนต์

2.4.2) $\det(AB) = \det(BA)$

2.4.3) $\det(AB) = \det(A)\det(B)$

2.4.4) $\det(A\backslash B) = \det(A)\backslash\det(B)$

2.4.5) $\det(A/B) = \det(A)/\det(B)$

2.4.6) $\det(A + B) \neq \det(A) + \det(B)$

2.4.7) $(A^{-1})^{-1} = A$ โดยที่ $(.)^{-1}$ คือ เครื่องหมายอินเวอร์สการคูณ

2.4.8) $(AB)^{-1} = B^{-1}A^{-1}$

2.4.9) $A/B = AB^{-1}$

2.4.10) $A\backslash B = A^{-1}B$

2.5 กำหนดให้

$$A = \begin{bmatrix} 1 & 1-2i \\ 1+2i & 1 \end{bmatrix} \qquad B = \begin{bmatrix} i & 2i \\ 3i & 4i \end{bmatrix}$$

เมื่อ $i = \sqrt{-1}$ จงคำนวณหาค่าต่อไปนี้พร้อมทั้งอธิบายผลลัพธ์ที่ได้

2.5.1) A^T โดยที่ $(.)^T$ คือ ทรานส์โพสเมทริกซ์แบบธรรมดา

2.5.2) A^H โดยที่ $(.)^H$ คือ ทรานส์โพสเมทริกซ์แบบสังยุค

2.5.3) $A^T + B^H$

2.5.4) $(AB)^H$

2.5.5) $B^H A^H$

2.6 จงหาสมการลักษณะเฉพาะของเมทริกซ์ต่อไปนี้ พร้อมทั้งหาคำตอบของสมการลักษณะเฉพาะที่ได้

2.6.1) $A = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$

2.6.2) $B = \begin{bmatrix} 1 & -5 \\ -2 & 3 \end{bmatrix}$

2.6.3) $C = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

2.6.4) $D = \begin{bmatrix} 2 & 1 & 3 \\ -2 & 0 & 2 \\ 5 & 3 & 1 \end{bmatrix}$

2.7 จงหาสร้างสมการพหุนามในรูปของตัวแปร x ที่มีรากหรือคำตอบของสมการดังนี้

2.7.1) $x = \{1, 2, 3\}$

2.7.2) $x = \{-3, 0, 2\}$

2.7.3) $x = \{2, 1 + i, 1 - i\}$

2.7.4) $x = \{-5, -3, -1, 0, 1, 3, 5\}$

2.7.5) $x = \{-3, 0, 3, -1 + 3i, -1 - 3i, 2 - 4i, 2 + 4i\}$

2.8 จงสร้างข้อมูลประเภท `list`, `tlist`, และ `mlist` พร้อมทั้งอธิบายข้อแตกต่างระหว่างข้อมูลแต่ละประเภท

บทที่ 3

ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการ (operator) หมายถึงตัวกระทำที่มีผลต่อค่าของข้อมูล ในทางปฏิบัติสามารถแบ่งตัวดำเนินการที่ใช้สำหรับการคำนวณทางคณิตศาสตร์ในโปรแกรม SCILAB ได้เป็นสามประเภท คือ ตัวดำเนินการเลขคณิต (arithmetic operator), ตัวดำเนินการสัมพันธ์และตรรกะ (relational and logical operators), และตัวดำเนินการระดับบิต (bit-wise operator) ในบทนี้จะอธิบายถึงลักษณะการทำงานของตัวดำเนินการแต่ละประเภท พร้อมทั้งแสดงตัวอย่างการใช้งานของตัวดำเนินการเหล่านี้ในการคำนวณทางคณิตศาสตร์

3.1 ตัวดำเนินการเลขคณิต

การทำงานของตัวดำเนินการจะเรียกว่าการดำเนินการ (operation) ในส่วนนี้จะอธิบายถึงการใช้งานตัวดำเนินการเลขคณิต (arithmetic operator) ลักษณะต่างๆ ดังต่อไปนี้

3.1.1 การดำเนินการทางคณิตศาสตร์ที่เกี่ยวข้องกับค่าคงที่

ในการคำนวณทางคณิตศาสตร์ระหว่างค่าสเกลาร์กับค่าสเกลาร์ เครื่องหมายที่ใช้ในการคำนวณกับเครื่องหมายที่ใช้ในโปรแกรม SCILAB จะต่างกันเล็กน้อย ดังที่แสดงในตารางที่ 3.1 ตัวอย่างเช่น ถ้าป้อนคำสั่ง `y = a + b` ลงในหน้าต่างคำสั่ง โปรแกรม SCILAB จะพิจารณาประโยคคำสั่งทางด้านขวามือของเครื่องหมายเท่ากับ “=” ก่อน แล้วจึงค่อยเอาผลลัพธ์ที่ได้ไปบรรจุไว้ในตัวแปรที่อยู่ทางด้านซ้ายมือของเครื่องหมายเท่ากับ “=” ซึ่งในที่นี้หมายความว่าให้นำค่า `a` มาบวกรวมกับค่า `b` แล้วเอาผลลัพธ์ที่ได้ไปบรรจุไว้ในตัวแปร `y` สำหรับตัวอย่างการคำนวณทางคณิตศาสตร์ระหว่างค่าสเกลาร์กับค่าสเกลาร์มีดังนี้

ตารางที่ 3.1 การดำเนินการที่ใช้ในการคำนวณทางคณิตศาสตร์ของค่าสเกลาร์

การดำเนินการ	รูปแบบพีชคณิต	รูปแบบของ SCILAB
การบวก (addition)	$a + b$	`a + b`
การลบ (subtraction)	$a - b$	`a - b`
การคูณ (multiplication)	$a \times b$	`a * b`
การหารซ้าย (left division)	$\frac{b}{a}$	`a \ b`
การหารขวา (right division)	$\frac{a}{b}$	`a / b`
การยกกำลัง (exponentiation)	a^b	`a^b` หรือ `a**b`

```
-->a = 3;

-->b = 2;

-->M = [a+b, a-b, a*b; a\b, a/b, a^b]
 M  =
    5.          1.     6.
    0.6666667   1.5    9.

-->y = [a^b, b^a, a**b, b**a]
 y  =
    9.    8.    9.    8.
```

3.1.2 ลำดับความสำคัญของการดำเนินการทางคณิตศาสตร์

จากตัวอย่างที่แสดงข้างต้นจะพบว่า การทำงานของตัวดำเนินการจะเรียงจากซ้ายไปขวาเสมอ เช่น `y = 3 – 2` จะมีผลลัพธ์เป็นค่า `1` (ไม่ใช่เป็นการนำค่า `3` มาลบออกจากค่า `2` ซึ่งจะให้ผลลัพธ์เป็น `-1`) อย่างไรก็ตามในกรณีที่ประโยคคำสั่งมีตัวดำเนินการหลายตัว การคำนวณทางคณิตศาสตร์จะต้องเป็นไปตามลำดับการทำงานของตัวดำเนินการ โดยจะต้องเป็นไปตามกฎการทำก่อน (precedence rule) เมื่อเข้าใจถึงลำดับของการดำเนินการแล้ว ก็สามารถที่จะเขียนคำสั่งบรรทัดเดียวเพื่อทำการคำนวณสมการคณิตศาสตร์ที่มีตัวดำเนินการหลายๆ ตัวได้ ตารางที่ 3.2 แสดงถึงลำดับการทำงานของตัวดำเนินการในโปรแกรม SCILAB ตัวอย่างเช่น

ตารางที่ 3.2 ลำดับการทำงานของตัวดำเนินการ

ลำดับความสำคัญ (priority)	ตัวดำเนินการ
1	วงเล็บ ( )
2	เลขยกกำลัง ^ หรือ ** และเรียงจากซ้ายไปขวา
3	การคูณและการหาร และเรียงจากซ้ายไปขวา
4	การบวกและการลบ และเรียงจากซ้ายไปขวา

```
-->y = 10 + 2 * 3 - 4 + 9 / 3 ^ 2 * 2 - 1
 y  =
    13.
-->y = 10 + (2 * 3) - 4 + (9 / (3 ^ 2)) * 2 - 1
 y  =
    13.
-->y = (10 + 2) * 3 - 4 + 9 / (3 ^ 2) * 2 - 1
 y  =
    33.
```

3.1.3 การดำเนินการทางคณิตศาสตร์ที่เกี่ยวข้องกับเมทริกซ์

ตัวดำเนินการที่ใช้ในการคำนวณทางคณิตศาสตร์สำหรับเมทริกซ์ แสดงในตารางที่ 3.3

3.1.3.1 การบวกและการลบ

การบวก (และการลบ) ของสองเมทริกซ์จะทำได้ก็ต่อเมื่อเมทริกซ์ทั้งสองมีขนาดเท่ากัน โดยผลลัพธ์ที่ได้จะเกิดจากการบวก (และการลบ) ของสมาชิกทีละตัวที่อยู่ ณ ตำแหน่งที่ตรงกันของทั้งสองเมทริกซ์ ตัวอย่างเช่น

```
-->A = [1 2 3; 4 5 6];
-->B = [1 1 1; -1 -1 -1];
-->A + B
 ans  =
    2.    3.    4.
    3.    4.    5.
```

ตารางที่ 3.3 ตัวดำเนินการที่ใช้ในการคำนวณทางคณิตศาสตร์ของเมทริกซ์

ตัวดำเนินการ	คำอธิบาย
+	การบวก (addition)
-	การลบ (subtraction)
*	การคูณ (multiplication)
.*	การคูณในระดับสมาชิก (element-wise multiplication)
.*.	การคูณแบบโครเนคเกอร์ (Kronecker product)
\	การหารซ้าย (left division)
.\	การหารซ้ายในระดับสมาชิก (element-wise left division)
.\.	การหารซ้ายแบบโครเนคเกอร์ (Kronecker left division)
/	การหารขวา (right division)
./	การหารขวาในระดับสมาชิก (element-wise right division)
./.	การหารขวาแบบโครเนคเกอร์ (Kronecker right division)
^ หรือ **	การยกกำลัง (exponentiation)
.^	การยกกำลังในระดับสมาชิก (element-wise exponentiation)

```
-->A - B
 ans  =
    0.    1.    2.
    5.    6.    7.
```

นอกจากนี้ยังสามารถทำการบวก (หรือการลบ) ตัวเลขที่เป็นสเกลาร์กับเมทริกซ์ได้ โดยผลลัพธ์ที่ได้จะมาจากการนำเอาตัวเลขที่เป็นสเกลาร์ไปบวก (หรือลบ) กับสมาชิกทุกตัวภายในเมทริกซ์นั้น ตัวอย่างเช่น

```
-->3 + B
 ans  =
    4.    4.    4.
    2.    2.    2.
```

3.1.3.2 การคูณ

การคูณของสองเมทริกซ์จะทำได้ก็ต่อเมื่อขนาดของสองเมทริกซ์นั้นสอดคล้องกับกฎเกณฑ์พื้นฐานของการคูณของเมทริกซ์ นั่นคือจำนวนแนวตั้งของตัวตั้งจะต้องเท่ากับจำนวนแถวของตัวคูณ เช่น

```
-->A = [1 2 3; 4 5 6];       //จำนวนแนวตั้งของเมทริกซ์ A เท่ากับ 3

-->B = [1 1 1; -1 -1 -1];    //จำนวนแถวของเมทริกซ์ B เท่ากับ 2

-->A * B                     //จำนวนแนวตั้งของตัวตั้งไม่เท่ากับจำนวนแถวของตัวคูณ
     --error    10
inconsistent multiplication

-->A * B'                    //โดยที่ B' คือทรานส์โพสของเมทริกซ์ B
 ans  =
    6.    - 6.
    15.   - 15.
```

เครื่องหมายการคูณอีกรูปแบบหนึ่งที่น่าสนใจคือ ".*" ซึ่งจะเรียกว่า "การคูณในระดับสมาชิก (element-wise multiplication)" การคูณแบบนี้จะเป็นการนำสมาชิกที่อยู่ในตำแหน่งเดียวกันของทั้งสองเมทริกซ์มาคูณกัน ดังนั้นการคูณแบบนี้จะเกิดขึ้นได้ก็ต่อเมื่อเมทริกซ์ทั้งสองจะต้องมีขนาดเท่ากัน ตัวอย่างเช่น

```
-->A .* B
 ans  =
    1.    2.    3.
  - 4.  - 5.  - 6.
```

เช่นเดียวกันผู้ใช้สามารถนำค่าสเกลาร์ไปคูณกับเมทริกซ์ได้โดยตรง ซึ่งผลลัพธ์ที่ได้จะมาจากการนำเอาค่าสเกลาร์ไปคูณกับสมาชิกทีละตัวในเมทริกซ์นั้น เช่น

```
-->2 * B                     //มีผลเท่ากับการใช้คำสั่ง 2.*B
 ans  =
    2.    2.    2.
  - 2.  - 2.  - 2.
```

3.1.3.3 การหาร

ในโปรแกรม SCILAB จะมีการหารอยู่สองรูปแบบคือ การหารซ้าย “\” และการหารขวา “/” ตัวอย่างการใช้งานของเครื่องหมายหารทั้งสองแบบมีดังนี้

```
-->A = [1 2; 3 4];
-->B = [1 1; -1 -2];
-->A/B                          //เมทริกซ์ A เป็นตัวตั้ง ส่วนเมทริกซ์ B เป็นตัวหาร
 ans  =
    0.   - 1.
    2.   - 1.
-->A*inv(B)                     //ให้ผลลัพธ์เหมือนกับการใช้คำสั่ง A/B
 ans  =
    0.   - 1.
    2.   - 1.
-->A\B                          //เมทริกซ์ B เป็นตัวตั้ง ส่วนเมทริกซ์ A เป็นตัวหาร
 ans  =
  - 3.   - 4.
    2.     2.5
-->inv(A)*B                     //ให้ผลลัพธ์เหมือนกับการใช้คำสั่ง A\B
 ans  =
  - 3.   - 4.
    2.     2.5
```

สำหรับการใช้เครื่องหมายหารซ้ายในระดับสมาชิก “.\” และเครื่องหมายการหารขวาในระดับสมาชิก “./” มีหลักการใช้งานเช่นเดียวกันกับเครื่องหมายการคูณในระดับสมาชิก “.*” กล่าวคือจะนำสมาชิกที่อยู่ในตำแหน่งเดียวกันของทั้งสองเมทริกซ์มาหารกัน ดังนั้นการหารในระดับสมาชิกจะเกิดขึ้นได้ก็ต่อเมื่อเมทริกซ์ทั้งสองมีขนาดเท่ากันเท่านั้น ตัวอย่างเช่น

```
-->A = [1 2; 3 4];
-->B = [2 1; 6 8];
-->A ./ B
 ans  =
```

```
    0.5    2.
    0.5    0.5

-->A .\ B
 ans  =
    2.     0.5
    2.     2.

-->2 .\ A                          //เอาค่า 2 ไปหารทุกสมาชิกในเมทริกซ์ A
 ans  =
    0.5    1.
    1.5    2.

-->2 ./ A                          //เท่ากับการใช้คำสั่ง A .\ 2
 ans  =
  - 4.     2.
    3.   - 1.
```

3.1.3.4 การคูณและการหารแบบโครเนคเกอร์

โปรแกรม SCILAB ได้เตรียมการคูณและการหารแบบโครเนคเกอร์ไว้ใช้งานกับเมทริกซ์ ซึ่งมีประโยชน์มากในการใช้งานทางด้านวิศวกรรม

การคูณแบบโครเนคเกอร์

เครื่องหมายที่ใช้สำหรับการคูณแบบโครเนคเกอร์ คือ ".*." หรือสามารถใช้คำสั่ง kron ก็ได้ โดยผลลัพธ์ที่ได้จะเป็นผลคูณแบบโครเนคเกอร์ของสองเมทริกซ์ กล่าวคือถ้าให้เมทริกซ์ A มีขนาด m×n โดยที่ A(i,j) คือสมาชิกของแถวที่ i และแนวตั้งที่ j และเมทริกซ์ B มีขนาด p×q ดังนั้น A.*.B จะมีค่าเท่ากับ

$$A.*.B = kron(A,B) = \begin{vmatrix} A(1,1)B & \cdots & A(1,n)B \\ \vdots & & \vdots \\ \vdots & & \vdots \\ \vdots & & \vdots \\ A(m,1)B & \cdots & A(m,n)B \end{vmatrix}$$

ซึ่งจะได้เป็นเมทริกซ์ใหม่ที่มีขนาดเท่ากับ (m*p)×(n*q) ตัวอย่างเช่น

```
-->A = [1 2; 3 4];

-->B = [1 1; -1 -1];

-->A.*.B
 ans  =
    1.    1.    2.    2.
  - 1.  - 1.  - 2.  - 2.
    3.    3.    4.    4.
  - 3.  - 3.  - 4.  - 4.

-->kron(A, B)                          //มีค่าเท่ากับการใช้คำสั่ง A.*.B
 ans  =
    1.    1.    2.    2.
  - 1.  - 1.  - 2.  - 2.
    3.    3.    4.    4.
  - 3.  - 3.  - 4.  - 4.
```

การหารแบบโครเนคเกอร์

นอกจากนี้ในโปรแกรม SCILAB ยังได้เตรียมการหารแบบโครเนคเกอร์ไว้ซึ่งมีทั้งแบบการหารซ้าย “.\.” และแบบการหารขวา “./.” สำหรับผู้ที่สนใจลองศึกษารายละเอียดการใช้งานเครื่องหมายการหารแบบโครเนคเกอร์ด้วยตนเองจากโปรแกรม SCILAB

3.1.3.5 การยกกำลัง

การใช้งานเครื่องหมายยกกำลังกับเมทริกซ์สามารถทำได้หลายรูปแบบ ดังแสดงในตัวอย่างต่อไปนี้

```
-->A = [1 2; 3 4];

-->A^2                                 //มีค่าเท่ากับการใช้คำสั่ง A*A
 ans  =
    7.    10.
    15.   22.

-->A**2                                //มีค่าเท่ากับการใช้คำสั่ง A^2
 ans  =
    7.    10.
    15.   22.
```

```
-->A.^2                          //การยกกำลังในระดับสมาชิก มีค่าเท่ากับ
 ans  =                          //[1^2  2^2;  3^2  4^2]
    1.    4.
    9.    16.
```

สังเกตว่าคำสั่ง A^2 คือการนำเอาเมทริกซ์ A มาคูณกันสองครั้ง นั่นคือ A*A ดังนั้นการใช้เครื่องหมายยกกำลังกับเมทริกซ์จึงใช้ได้กับเมทริกซ์จัตุรัสเท่านั้น ในขณะที่คำสั่ง A.^2 คือการยกกำลังในระดับสมาชิกกับค่าสเกลาร์ ซึ่งในที่นี้หมายถึงให้ทำการยกกำลังสองให้กับสมาชิกแต่ละตัวในเมทริกซ์ A ฉะนั้นการยกกำลังในระดับสมาชิกกับค่าสเกลาร์สามารถใช้กับเมทริกซ์ขนาดใดก็ได้

นอกจากนี้เครื่องหมายยกกำลังกับเมทริกซ์ยังสามารถใช้งานในรูปแบบอื่นได้อีก เช่น

```
-->2.^A                          //คำสั่งนี้มีผลเทียบเท่ากับการใช้คำสั่ง 2^A
 ans  =
    2.    4.
    8.    16.
-->B = [1 1; -1 -1];
-->A.^B                          //มีค่าเท่ากับ [1^2, 2^1; 3^(-1), 4^(-1)]
 ans  =
    1.           2.
    0.3333333    0.25
-->A^B                           //การนำเอาเมทริกซ์ A มายกกำลังเมทริกซ์ B ไม่สื่อความหมายใดๆ
  --error    43
not implemented in scilab....
at line      60 of function %s_pow called by :
A^B
```

3.1.4 การแก้ระบบสมการเชิงเส้น

ในส่วนนี้จะแสดงการประยุกต์ใช้งานเมทริกซ์ในการแก้ไขปัญหาระบบสมการเชิงเส้น (linear equation system) ถ้าต้องการแก้สมการสองตัวแปรเพื่อหาค่าของตัวแปร x_1 และ x_2 จาก

$$2x_1 + x_2 = 3 \quad (1)$$

$$x_1 - x_2 = 3 \quad (2)$$

ถ้าใช้หลักการแก้สมการคณิตศาสตร์สองตัวแปรทั่วไปเพื่อแก้สมการที่ (1) และ (2) จะได้ผลลัพธ์คือ $x_1 = 2$ และ $x_2 = -1$ เช่นเดียวกันผู้ใช้สามารถแก้สมการทั้งสองนี้ได้โดยใช้เมทริกซ์ดังนี้

จากสมการที่ (1) และ (2) สามารถเขียนสมการทั้งสองให้อยู่ในรูปของเมทริกซ์ได้ คือ

$$\begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \quad \text{หรือ} \quad Ax = b \tag{3}$$

โดยที่ $A = \begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix}$ และ $b = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$ การหาผลเฉลยของสมการที่ (3) ทำได้โดยนำ A^{-1} (อินเวอส์การคูณของเมทริกซ์ A) คูณเข้าไปทั้งสองข้างของสมการที่ (3) ซึ่งจะได้ผลลัพธ์เป็น

$$A^{-1}Ax = A^{-1}b$$

$$x = A^{-1}b \tag{4}$$

เนื่องจาก $A^{-1}A = I$ คือเมทริกซ์เอกลักษณ์ (identity matrix) ถ้าใช้โปรแกรม SCILAB ในการหาผลเฉลยของสมการที่ (1) และ (2) ก็สามารถทำได้โดยใช้ชุดคำสั่งดังนี้

```
-->A = [2 1; 1 -1];                    //เมทริกซ์ขนาด 2×2
-->b = [3; 3];                         //เวกเตอร์แนวตั้งขนาด 2×1
-->x = inv(A)*b                        //คำตอบของสมการสองตัวแปร
 x  =                                  //เป็นเวกเตอร์แนวตั้งขนาด 2×1
    2.
  - 1.
-->x = A\b                   //เทียบเท่ากับการใช้คำสั่ง x = inv(A)*b
 x  =
    2.
  - 1.
```

ผลลัพธ์ที่ได้คือ $x_1 = 2$ และ $x_2 = -1$ ดังนั้นจะเห็นได้ว่าเครื่องหมายหารซ้าย “\” สามารถนำมาใช้ในการหาผลเฉลยของสมการที่อยู่ในรูป $Ax = b$ ได้

นอกจากนี้โปรแกรม SCILAB ยังได้เตรียมคำสั่งสำหรับการแก้ปัญหาระบบสมการเชิงเส้นแบบหลายตัวแปรไว้แล้ว นั่นคือคำสั่ง `linsolve` ซึ่งใช้ในการหาคำตอบของสมการหลายตัวแปรที่สามารถจัดให้อยู่ในรูปของเมทริกซ์

$$Ax + b = 0 \tag{5}$$

รูปแบบการใช้งานของคำสั่ง `linsolve` คือ

```
x = linsolve(A, b)
```

โดยที่พารามิเตอร์ `x`, `A`, และ `b` ทำหน้าที่เหมือนกับที่กล่าวมาแล้วข้างต้น ตัวอย่างการใช้งานของคำสั่งนี้ เช่น

```
-->A = [2 1; 1 -1];
-->b = [3; 3];
-->x = linsolve(A, -b)      //เท่ากับการใช้คำสั่ง x = inv(A)*b หรือ x = A\b
 x  =
    2.
  - 1.
```

สำหรับการหาค่าผลเฉลย `x` ของสมการที่อยู่ในรูปของ

$$xA = b \tag{6}$$

สามารถทำได้โดยการนำ A^{-1} คูณเข้าไปทั้งสองข้างของสมการที่ (6) ซึ่งจะได้ผลลัพธ์เป็น

$$xAA^{-1} = bA^{-1}$$

$$x = bA^{-1} \tag{7}$$

ตัวอย่างเช่นจากสมการที่ (1) และ (2) สามารถเขียนความสัมพันธ์ของเมทริกซ์ในอีกรูปแบบหนึ่งที่แตกต่างจากสมการที่ (3) ได้คือ

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \end{bmatrix} \quad (8)$$

ดังนั้นผลเฉลยของสมการที่ (8) สามารถหาได้โดยการใช้ชุดคำสั่งในโปรแกรม SCILAB ดังนี้

```
-->A = [2 1; 1 -1];
-->b = [3 3];                    //เวกเตอร์แถวขนาด 1×2
-->x = b*inv(A)                  //เทียบเท่ากับการใช้คำสั่ง x = b/A
 x  =
    2.   - 1.
-->x = b/A
 x  =
    2.   - 1.                    //เป็นเวกเตอร์แถวขนาด 1×2
```

ผลลัพธ์ที่ได้คือ $x_1 = 2$ และ $x_2 = -1$ ดังนั้นเครื่องหมายหารขวา “/” สามารถนำมาใช้ในการหาผลเฉลยของสมการที่อยู่ในรูป $xA = b$ ได้

3.1.5 การดำเนินการทางคณิตศาสตร์ที่เกี่ยวข้องกับพหุนาม

โปรแกรม SCILAB สามารถที่จะทำการดำเนินการทางคณิตศาสตร์กับพหุนามโดยตรงได้ทันที ซึ่งเป็นประโยชน์มากในการใช้งานทางด้านวิศวกรรม (โดยเฉพาะอย่างยิ่งทางด้านวิศวกรรมระบบควบคุม) ตัวอย่างการดำเนินการต่างๆ มีดังต่อไปนี้

```
-->p = poly([1 2], 's')          //สร้างสมการพหุนามจากคำตอบของสมการพหุนาม
 p  =                            //นั่นคือคำตอบของสมการพหุนาม p คือ s = 1 และ s = 2
                2
    2 - 3s + s
-->q = poly([1 2], 's', 'c')     //สร้างสมการพหุนามจากค่าสัมประสิทธิ์
 q  =
    1 + 2s
-->X = [p+q, p-q, p*q; p/q, p\q, 0]
```

```
X  =
                2                  2                  2    3
   3 - s + s       1 - 5s + s      2 + s - 5s + 2s
   ---------       ----------      ---------------
       1               1                  1

              2
   2 - 3s + s        1 + 2s        0
   ----------      ----------      -
                                   2
     1 + 2s        2 - 3s + s      1
```

ถ้ากำหนดให้ตัวแปรที่ใช้ในสมการพหุนามมีค่าเป็นเลขจำนวนจริงหรือเลขจำนวนเชิงซ้อน ผลลัพธ์ของสมการพหุนามสามารถหาค่าได้จากการใช้คำสั่ง `horner` ซึ่งมีรูปแบบการใช้งาน คือ

```
horner(P, y)
```

โดยที่พารามิเตอร์

- `P` คือ สมการพหุนามที่เป็นฟังก์ชันของตัวแปร `x`
- `y` คือ ค่าจำนวนจริงของตัวแปร `x`

ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->s = poly(0, 's');
-->A = [1, s; -s, 1+s^2]
 A  =
    1      s
                2
  - s      1 + s

-->B = horner(A, 2)          //แทนค่า s = 2 ลงในเมทริกซ์ A ก็จะได้ผลลัพธ์เป็นเมทริกซ์ B
 B  =
    1.     2.
  - 2.     5.

-->C = horner(A, 3+4*%i)
```

```
 C  =
    1.           3. + 4.i
  - 3. - 4.i   - 6. + 24.i
```

นอกจากนี้โปรแกรม SCILAB ยังได้เตรียมชุดคำสั่งอื่นๆ ที่เกี่ยวข้องกับการประมวลผลของพหุนามไว้จำนวนมาก ลองใช้คำสั่ง `help polynomial` เพื่อดูลักษณะการใช้งานของคำสั่งเหล่านั้น

3.2 ตัวดำเนินการสัมพันธ์และตรรกะ

ตัวดำเนินการสัมพันธ์และตรรกะ (relational and logical operator) จะใช้ในการเปรียบเทียบความสัมพันธ์ของค่าของตัวแปรทั้งที่เป็นสเกลาร์และเมทริกซ์ โดยตัวแปรทั้งสองที่ใช้ในการเปรียบเทียบจะต้องมีขนาด (หรือมิติ) เท่ากัน ในกรณีที่ตัวแปรที่ต้องการเปรียบเทียบเป็นเมทริกซ์ การเปรียบเทียบจะกระทำในระดับสมาชิกแบบหนึ่งต่อหนึ่ง

3.2.1 ตัวดำเนินการสัมพันธ์

ตัวดำเนินการสัมพันธ์ (relational operator) เป็นตัวดำเนินการที่ใช้ในการตรวจสอบความสัมพันธ์ระหว่างค่าของตัวแปร โดยที่

- ถ้าความสัมพันธ์เป็นจริง ผลลัพธ์ที่ได้จะมีค่าเท่ากับ `T` (True) หรือมีค่าทางตรรกะเท่ากับค่า `1`
- ถ้าความสัมพันธ์เป็นเท็จ ผลลัพธ์ที่ได้จะมีค่าเท่ากับ `F` (False) หรือมีค่าทางตรรกะเท่ากับค่า `0`

โปรแกรม SCILAB ได้เตรียมตัวดำเนินการที่ใช้ตรวจสอบความสัมพันธ์ไว้ทั้งหมดหกแบบ ตามที่แสดงในตารางที่ 3.4 ตัวอย่างการใช้งาน เช่น

```
-->x = [1 2 3];
-->y = [-1 2 5];
-->x == y                    //เป็นการเปรียบเทียบค่าในแต่ละสมาชิก
 ans  =
  F T F
-->x < y
 ans  =
  F F T
```

ตารางที่ 3.4 ตัวดำเนินการสัมพันธ์ (relational operator)

ตัวดำเนินการสัมพันธ์	คำอธิบาย
==	เครื่องหมายเท่ากับ (equal to)
<	เครื่องหมายน้อยกว่า (less than)
>	เครื่องหมายมากกว่า (greater than)
<=	เครื่องหมายน้อยกว่าหรือเท่ากับ (less than or equal to)
>=	เครื่องหมายมากกว่าหรือเท่ากับ (greater than or equal to)
<> หรือ ~=	เครื่องหมายไม่เท่ากับ (not equal to)

```
-->x + y > 5                    //เนื่องจาก x + y = [0 4 8]
 ans  =
  F F T
-->x .* y <= 4                  //เนื่องจาก x.*y = [-1  4  15]
 ans  =
  T T F
-->x >= y/2                     //เนื่องจาก y/2 = [-0.5  1  2.5]
 ans  =
  T T T
-->x ~= y                       //เหมือนกับการใช้คำสั่ง x <> y
 ans  =
  T F T
```

3.2.2 ตัวดำเนินการตรรกะ

ตัวดำเนินการตรรกะ (logical operator) เป็นตัวดำเนินการที่ใช้เชื่อมความสัมพันธ์ระหว่างค่าของตัวแปรที่เกิดขึ้นโดยตัวดำเนินการสัมพันธ์ โดยที่

- ถ้าความสัมพันธ์สอดคล้องกันผลลัพธ์ที่ได้ก็จะมีค่าเท่ากับ T (เป็นจริง) หรือมีค่าทางตรรกะเท่ากับค่า 1

ตารางที่ 3.5 ตัวดำเนินการตรรกะ (logical operator)

ตัวดำเนินการตรรกะ	คำอธิบาย
&	และ (AND)
|	หรือ (OR)
~	ไม่ (NOT)

P AND *Q*

Q \ *P*	T	F
T	T	F
F	F	F

P OR *Q*

Q \ *P*	T	F
T	T	T
F	T	F

NOT *P*

P	ผลลัพธ์
T	F
F	T

รูปที่ 3.1 ผลลัพธ์ที่ได้จากการดำเนินการของตัวดำเนินการตรรกะ

- ถ้าความสัมพันธ์ไม่สอดคล้องกันผลลัพธ์ที่ได้ก็จะมีค่าเท่ากับ F (เป็นเท็จ) หรือมีค่าทางตรรกะเท่ากับค่า 0

ตัวดำเนินการตรรกะในโปรแกรม SCILAB มีอยู่สามแบบตามตารางที่ 3.5 โดยที่ผลลัพธ์ที่ได้จากการใช้ตัวดำเนินการตรรกะเป็นไปตามรูปที่ 3.1 ตัวอย่างเช่น

```
-->P = [%t %t %f %f]          //กำหนดให้ P เป็นเวกเตอร์ตรรกะ (logical vector)
 P  =
  T T F F

-->Q = [%t %f %t %f]          //กำหนดให้ Q เป็นเวกเตอร์ตรรกะ
 Q  =
  T F T F

-->P & Q                      //ได้ผลลัพธ์เหมือนตาราง P AND Q ในรูปที่ 3.1
 ans  =
  T F F F
```

```
-->P | Q                          //ได้ผลลัพธ์เหมือนตาราง P OR Q ในรูปที่ 3.1
 ans  =
  T T T F

-->~P                            //ได้ผลลัพธ์เหมือนตาราง NOT P ในรูปที่ 3.1
 ans  =
  F F T T

-->P + 1                         //เวกเตอร์ P เปรียบเสมือนมีค่าเท่ากับ P = [1 1 0 0]
 ans  =
    2.    2.    1.    1.

-->2 * Q                         //เวกเตอร์ Q เปรียบเสมือนมีค่าเท่ากับ Q = [1 0 1 0]
 ans  =
    2.    0.    2.    0.

-->P + Q                         //ตัวแปรบูลีน T จะเปรียบเสมือนมีค่าเท่ากับค่า 1
 ans  =                          //และตัวแปรบูลีน F จะเปรียบเสมือนมีค่าเท่ากับค่า 0
    2.    1.    1.    0.
```

จะเห็นได้ว่าเมื่อนำเวกเตอร์ตรรกะมาดำเนินการทางคณิตศาสตร์ (เช่น การบวก, การลบ, การคูณ, และการหาร) กับค่าสเกลาร์แล้ว เวกเตอร์ตรรกะสามารถที่จะแทนได้ด้วยเวกเตอร์ของค่าสเกลาร์ที่มีสมาชิกเป็นค่า 0 กับค่า 1 โดยที่ค่า 0 จะใช้แทนข้อความ (statement) ที่เป็นเท็จ และค่า 1 จะใช้แทนข้อความที่เป็นจริง (ศึกษาตัวอย่างการประยุกต์ใช้งานเวกเตอร์ตรรกะเพิ่มเติมได้ในบทที่ 9)

นอกจากนี้ตัวดำเนินการตรรกะยังสามารถนำมาใช้เชื่อมรวมการทดสอบความสัมพันธ์ทางตรรกะหลายๆ เงื่อนไขเข้าด้วยกันได้ โดยผลลัพธ์ที่ได้จะเป็นไปตามลำดับความสำคัญของตัวดำเนินการต่างๆ ตามที่แสดงในตารางที่ 3.6 ตัวอย่างเช่น

```
-->x = 1;

-->y = 2;

-->(x + 2 * y < 1) & (4 / 2 == 1)
 ans  =
  F

-->x > 5 & y == 10               //เหมือนกับการใช้ (x > 5) & (y ==10)
 ans  =
  F
```

ตารางที่ 3.6 ลำดับการทำงานของตัวดำเนินการต่างๆ

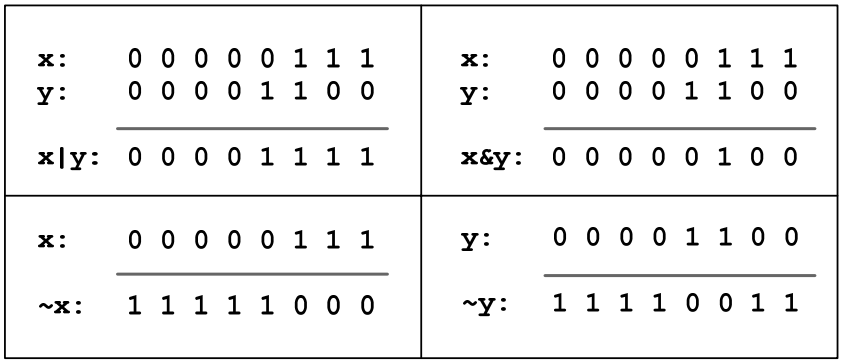
ลำดับความสำคัญ (priority)	ตัวดำเนินการ	สัญลักษณ์ตัวดำเนินการ
1	วงเล็บ	( )
2	ปฏิเสธ	~
3	เลขคณิต	+ - * /
4	สัมพันธ์	== < > <= >= ~=
5	ตรรกะ	& |

```
-->y * x < 1 | 2 + y > 5    //เหมือนกับการใช้ ((y*x)<1) | ((2+y)>5)
 ans  =
  F
-->x + 1 >= y - 2 & 2 * x == 1
 ans  =                     //เหมือนกับการใช้ ((x+1)>=(y-2))&((2*x)==1)
  F
```

3.3 ตัวดำเนินการระดับบิต

ตัวดำเนินการระดับบิต (bit-wise operator) เป็นตัวดำเนินการที่ทำงานในระดับบิต นั่นคือเป็นการนำข้อมูลของตัวแปรแต่ละตัวที่เก็บอยู่ในรูปของเลขฐานสอง (binary number) มาทำการคำนวณกัน ในโปรแกรม SCILAB ตัวดำเนินการระดับบิตจะเป็นตัวเดียวกันกับตัวดำเนินการตรรกะ เพียงแต่ข้อมูลที่จะนำมาดำเนินการในระดับบิตจะต้องเป็นเลขจำนวนเต็มเท่านั้น มิฉะนั้นแล้วโปรแกรม SCILAB จะไม่ทราบว่าให้นำข้อมูลเหล่านั้นมาดำเนินการในระดับบิต ให้พิจารณาตัวอย่างต่อไปนี้จะได้เข้าใจถึงหลักการทำงานของตัวดำเนินการระดับบิต

```
-->x = uint8(7)        //กำหนดให้ตัวแปร x เป็นเลขจำนวนเต็มขนาด 8 บิต ที่มีค่าเท่ากับค่า 7
 x  =
  7                    //ไม่มีจุดทศนิยมตามหลังตัวเลข แสดงว่าเป็นเลขจำนวนเต็ม
-->y = uint8(12)       //กำหนดให้ตัวแปร y เป็นเลขจำนวนเต็มขนาด 8 บิตที่มีค่าเท่ากับค่า 12
 y  =
  12
```



รูปที่ 3.2 ตัวอย่างการดำเนินการในระดับบิตของตัวแปร x และ y

```
-->[x|y, x&y, ~x, ~y]
 ans  =
  15   4   248   243
```

ผลลัพธ์ที่ได้คือการนำค่าตัวแปร x และ y มาดำเนินการในระดับบิต ตามที่แสดงในรูปที่ 3.2

หมายเหตุ ถ้าตัวแปร x และ y ไม่ได้เป็นเลขจำนวนเต็ม ผลลัพธ์ที่ได้จะเปรียบเสมือนกับการใช้ตัวดำเนินการตรรกะ ตัวอย่างเช่น

```
-->x = 7
 x  =
    7.                  //ตัวแปร x เป็นเลขจำนวนจริง (สังเกตจาก มีจุดทศนิยมตามหลังตัวเลข)

-->y = 12
 y  =
    12.                 //ตัวแปร y เป็นเลขจำนวนจริง

-->[x|y, x&y, ~x, ~y]
 ans  =
  T T F F
```

ผลลัพธ์ที่ได้คือค่าแสดงความสัมพันธ์ว่าเป็นจริงหรือเป็นเท็จ ทั้งนี้เนื่องจากตัวแปร x และ y ไม่ได้เป็นเลขจำนวนเต็ม สาเหตุที่ได้ผลลัพธ์เป็น [T T F F] เนื่องมาจากโปรแกรม SCILAB จะถือว่าค่าคงที่ต่างๆ ที่นำมาใช้ในการทดสอบหาความสัมพันธ์จะมีค่าเป็นจริง (เท่ากับ T) เสมอ เช่น

```
-->x = -1;

-->[x|2, x&3, ~4, ~x]
 ans  =
  T T F F
```

3.4 ตัวอย่างการคำนวณ

ในส่วนนี้จะยกตัวอย่างการแก้ไขปัญหาโจทย์ทางคณิตศาสตร์โดยใช้โปรแกรม SCILAB ดังต่อไปนี้

ตัวอย่างที่ 1 จงหาค่าของฟังก์ชัน $f(x) = \frac{3x^3-x}{2x^2+1} + 5x - 4$ เมื่อ x มีค่าเท่ากับ `-3i, -3, 0, 3` และ `3i` โดยที่ $i = \sqrt{-1}$

วิธีทำ คำตอบของโจทย์ข้อนี้หาได้จากการใช้ชุดคำสั่งดังนี้

```
-->x = [-3*%i, -3, 0, 3, 3*%i];
-->fx = (3*x^3-x)/(2*x^2+1) + 5*x - 4
 fx  =
  - 4. - 15.i   - 19.   - 4.    11.   - 4. + 15.i
```

นั่นคือ `f(x)` มีค่าเท่ากับ `-4-15i, -19, -4, 11,` และ `-4+15i` เมื่อ x มีค่าเท่ากับ `-3i, -3, 0, 3` และ `3i`

ตัวอย่างที่ 2 จงหาค่าของ $x = 2^{2^{2^2}} + (-2)^{2^{(-2)^2}}$

วิธีทำ ค่าของ x หาได้จาก

```
-->x = 2^2^2^2 + (-2)^(2)^(-2)^(2)
 x  =
    131072.
```

ตัวอย่างที่ 3 จงหาค่าของ $x = \left(\frac{2^3 \times 3^{-4}}{2^{-2} \times 3}\right)^{-3} \div \left(\frac{2^{-3} \times 3^2}{2 \times 3^{-1}}\right)^5$

วิธีทำ ค่าของ x หาได้จาก

```
-->num = ((2^3*3^(-4))/(2^(-2)*3))^(-3);

-->den = ((2^(-3)*3^2)/(2*3^(-1)))^5;

-->x = num/den
 x  =
    32.
```

ตัวอย่างที่ 4 จงหาค่าของตัวแปร u และ v จากระบบสมการเชิงเส้น

$$2u + 3v = 16$$

$$3u - 4v = 41$$

วิธีทำ จัดรูปสมการทั้งสองให้อยู่ในรูปของเมทริกซ์ Ax = b จากนั้นก็จะได้ว่าคำตอบของระบบสมการเชิงเส้นนี้คือ $x = A^{-1}b$ ในโปรแกรม SCILAB สามารถแก้สมการได้ดังนี้

```
-->A = [2, 3;  3, -4];

-->b = [16; 41];

-->x = inv(A)*b
 x  =
    11.
  - 2.
```

ผลลัพธ์ที่ได้คือ u = 11 และ v = -2 เป็นคำตอบของระบบสมการเชิงเส้นนี้

ตัวอย่างที่ 5 จงหาค่าของตัวแปร u, v, และ w จากระบบสมการเชิงเส้น

$$u + v + w = 2$$

$$2u - v - 2w = -1$$

$$-u + 2v + 2w = 1$$

วิธีทำ จัดรูปสมการทั้งสองให้อยู่ในรูปของเมทริกซ์ $Ax = b$ จากนั้นก็จะได้ว่าคำตอบของระบบสมการเชิงเส้นนี้คือ $x = A^{-1}b$ ในโปรแกรม SCILAB สามารถแก้สมการได้ดังนี้

```
-->A = [1, 1, 1;  2, -1, -2;  -1, 2, 2];

-->b = [2; -1; 1];

-->x = inv(A)*b
 x  =
    1.
  - 1.
    2.
```

ผลลัพธ์ที่ได้คือ $u = 1, v = -1,$ และ $w = 2$ เป็นคำตอบของระบบสมการเชิงเส้นนี้

ตัวอย่างที่ 6 กำหนดให้พหุนาม $p = x^2 - 1$ และ $q = 2x + 3$ จงหาค่าของเมทริกซ์

$$M = \begin{bmatrix} p+q & p-q & pq \\ \frac{p}{q} & \frac{q}{p} & \frac{p^2-2q}{3p+q} \end{bmatrix}$$

สำหรับ $x = -2$ และ $x = 1 + i$

วิธีทำ คำตอบของโจทย์ข้อนี้สามารถหาได้จากชุดคำสั่งดังนี้

```
-->x = poly(0, 'x');

-->p = x^2 - 1;

-->q = 2*x + 3;
```

```
-->M = [p+q, p-q, p*q;  p/q, q/p, (p^2-2*q)/(3*p+q)]
 M  =
                2               2                2    3
    2 + 2x + x    - 4 - 2x + x    - 3 - 2x + 3x + 2x
    ----------      ----------      ----------------
        1               1                 1

          2                                   2   4
  - 1 + x           3 + 2x        - 5 - 4x - 2x + x
    -----           ------          ---------------
                         2                   2
    3 + 2x        - 1 + x               2x + 3x

-->y = horner(M, -2)
 y  =
    2.     4.          - 3.
  - 3.   - 0.3333333    1.375

-->y = horner(M, 1+%i)
 y  =
    4. + 4.i                 - 6.           - 9. + 8.i
  - 0.0344828 + 0.4137931i   - 0.2 - 2.4i   - 1.3235294 + 1.2941176i
```

3.5 สรุป

ในบทนี้ได้อธิบายถึงลักษณะการใช้งานของตัวดำเนินการแบบต่างๆ ได้แก่ ตัวดำเนินการเลขคณิต, ตัวดำเนินการสัมพันธ์, ตัวดำเนินการตรรกะ, และตัวดำเนินการระดับบิต พร้อมทั้งแสดงตัวอย่างการใช้งานตัวดำเนินการต่างๆ ในการแก้ไขปัญหาโจทย์ทางคณิตศาสตร์ เมื่อเข้าใจถึงลักษณะการทำงานของตัวดำเนินการเหล่านี้แล้ว ก็จะช่วยทำให้สามารถนำตัวดำเนินการเหล่านี้ไปใช้ในการพัฒนาโปรแกรมที่ซับซ้อนได้อย่างถูกต้องตามที่ต้องการ

3.6 แบบฝึกหัดท้ายบท

3.1 จงคำนวณหาค่าของฟังก์ชันต่อไปนี้

3.1.1) $f(x)=3x^3+2x^2+1$ เมื่อ x มีค่าเท่ากับ –10, –5, 5, และ 10

3.1.2) $f(x)=\frac{2x^2+1}{x^2-3}+5x$ เมื่อ x มีค่าเท่ากับ –5i, –5, 0, 5 และ 5i โดยที่ $i=\sqrt{-1}$

3.2 จงอธิบายลำดับของการดำเนินการในการคำนวณของสมการต่อไปนี้

3.2.1) y = 1 + 2 * 3 – 4

3.2.2) y = 1 + 2 – 3 * 4 / 5

3.2.3) y = 1 + 2 – 3 * 4 / 5 \ 6 ^ 7

3.2.4) y = 1 * 2 + 3 ^ 4 \ 5 – 6 / 7 ** 8

3.2.5) y = (1 + 2) * 3 \ 4 – 5 ^ (6 – 2) / 3 ** 4

3.2.6) y = 1 – 2 \ (3 + 4) * 5 / (6 – 1) ** 2 – 2 ^ (3 – 1) * 2

3.3. กำหนดให้

$$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \qquad B=\begin{bmatrix}-1 & 2\\ -2 & 3\end{bmatrix}$$

จงคำนวณหาค่าต่อไปนี้ พร้อมทั้งอธิบายผลลัพธ์ที่ได้

3.3.1) 2 ^ A

3.3.2) 2**A

3.3.3) A*A

3.3.4) 2 .^ A

3.3.5) A .^ 2

3.3.6) A .* B

3.3.7) A .*. B

3.3.8) A .\ B

3.3.9) A ./ B

3.3.10) A .^ B

3.4. จงแก้สมการเพื่อหาค่าของตัวแปรทั้งหมดในระบบสมการเชิงเส้น (linear equation system) ต่อไปนี้

3.4.1) $x + y = 5$

$2x - y = 1$

3.4.2) $2x - 3y = 5$

$-6x + 9y = -12$

3.4.3) $2x - 3y = -5$

$-3x + 2y = 1$

3.4.4) $x + y + z = 6$

$x - 2y + z = 0$

$3x + y - 2z = -1$

3.4.5) $2x - 2y + z = 1$

$-x + y - z = -2$

$-2x + y - 2z = -6$

3.4.6) $x + 2y + 3z = 14$

$5x + y - 2z = 1$

$2x - 3y + z = -1$

3.5. กำหนดให้

$$x = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \qquad y = \begin{bmatrix} -4 & 2 & 4 \end{bmatrix}$$

จงคำนวณหาค่าต่อไปนี้ พร้อมทั้งอธิบายผลลัพธ์ที่ได้

3.5.1) x < y

3.5.2) x + y >= 4

3.5.3) x == y/2

3.5.4) x .^ y < 5

3.5.5) $x^T y < 5$ โดยที่ $(.)^T$ คือ ทรานส์โพสของเมทริกซ์แบบธรรมดา

3.5.6) $x^Ty <> y^Tx$

3.6. กำหนดให้ x = 2 และ y = 3 จงคำนวณหาค่าต่อไปนี้ พร้อมทั้งอธิบายผลลัพธ์ที่ได้

3.6.1) x * y < 2 | 3

3.6.2) x > 3 & y == 5

3.6.3) (x + 2 * y > 3) & ~ (6 | 3 == 2)

3.6.4) x + 1 >= y – 3 & 2 * x == 2

3.6.5) ~ (x > 2) & (y – 3 == 2) | x * y <= 1

3.7. กำหนดให้ x = 121 และ y = 19 เป็นเลขจำนวนเต็ม จงคำนวณหาค่าต่อไปนี้ พร้อมทั้งอธิบายผลลัพธ์ที่ได้

3.7.1) x | y

3.7.2) x & y

3.7.3) ~ x

3.7.4) ~ y

3.7.5) x & y | ~ y

บทที่ 4

ฟังก์ชันพื้นฐานทางคณิตศาสตร์

ในบทนี้จะอธิบายถึงฟังก์ชันพื้นฐานทั่วไปที่ใช้ในการคำนวณทางคณิตศาสตร์ เช่น ฟังก์ชันพื้นฐานที่เกี่ยวกับตัวเลข, ฟังก์ชันตรีโกณมิติ, ฟังก์ชันไฮเพอร์โบลิก, ฟังก์ชันพื้นฐานทางสถิติ รวมทั้งฟังก์ชันสำหรับการสร้างเมทริกซ์พิเศษ เพื่อให้เข้าใจถึงรูปแบบการใช้งานของฟังก์ชันต่างๆ ที่จำเป็นสำหรับการพัฒนาโปรแกรม

4.1 ฟังก์ชันพื้นฐานที่เกี่ยวกับตัวเลข

ในส่วนนี้จะกล่าวถึงฟังก์ชันพื้นฐานที่เกี่ยวกับตัวเลข (elementary number function) ที่พบมากในโปรแกรม SCILAB ตามที่แสดงในตารางที่ 4.1 ซึ่งมีลักษณะการใช้งานดังต่อไปนี้

4.1.1 ฟังก์ชัน `abs(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่าสัมบูรณ์ (absolute value) ของตัวแปร `x` ซึ่งแบ่งได้เป็น 2 กรณี คือ

1) ถ้าตัวแปร `x` เป็นเลขจำนวนจริง ค่าสัมบูรณ์ของตัวแปร `x` จะมีค่าเท่ากับ

$$|x| = \begin{cases} x & ,x \geq 0 \\ -x & ,x < 0 \end{cases}$$

2) ถ้าตัวแปร `x` เป็นเลขจำนวนเชิงซ้อนที่สามารถเขียนให้อยู่ในรูปของ `x = a + bi` โดยที่ตัวแปร `a` และ `b` เป็นเลขจำนวนจริง และ `i` = $\sqrt{-1}$ เป็นหน่วยจินตภาพ ดังนั้นค่าสัมบูรณ์ของตัวแปร `x` จะมีค่าเท่ากับ

$$|x| = \sqrt{a^2 + b^2}$$

ตารางที่ 4.1 ตัวอย่างฟังก์ชันพื้นฐานที่เกี่ยวกับตัวเลข

ฟังก์ชัน	คำอธิบาย
`abs(x)`	หาค่าสัมบูรณ์ (absolute value) ของตัวแปร `x`
`sqrt(x)`	หาค่ารากที่สอง (square root) ของตัวแปร `x`
`int(x)`	หาค่าจำนวนเต็มของตัวแปร `x`
`modulo(m,n)`	หาค่าเศษที่เหลือการหารตัวแปร `n` ด้วย `m`
`ceil(x)`	หาค่าจำนวนเต็มที่มีค่าใกล้กับค่า `x` ไปทางค่า ∞ มากที่สุด
`floor(x)`	หาค่าจำนวนเต็มที่มีค่าใกล้กับค่า `x` ไปทางค่า $-\infty$ มากที่สุด
`round(x)`	หาค่าจำนวนเต็มที่มีค่าใกล้กับค่า `x` มากที่สุด
`fix(x)`	หาค่าจำนวนเต็มที่มีค่าใกล้กับค่า `x` ไปทางค่า `0` มากที่สุด
`rat(x)`	ประมาณค่าจำนวนจริงให้อยู่ในรูปของเลขเศษส่วน
`sign(x)`	หาค่าเครื่องหมายของตัวแปร `x`
`roots(p)`	หาค่ารากหรือคำตอบของสมการพหุนาม `p`
`real(x)`	หาค่าจำนวนจริงของตัวแปร `x`
`imag(x)`	หาค่าจำนวนจินตภาพของตัวแปร `x`
`conj(x)`	หาค่าสังยุคของจำนวนจำนวเชิงซ้อนของตัวแปร `x`

ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->abs([1, %i, -2, -2*%i, 3+4*%i])        //หาค่าสัมบูรณ์ของแต่ละสมาชิกในเวกเตอร์
 ans  =
    1.    1.    2.    2.    5.
```

4.1.2 ฟังก์ชัน `sqrt(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่ารากที่สองของตัวแปร `x` เช่น ถ้า `x = y*y` โดยที่ `y` เป็นเลขจำนวนจริง ดังนั้น `sqrt(x)` = $|y|$ ในกรณีที่ตัวแปร `x < 0` จะได้ว่า

$$\sqrt{x} = i\sqrt{-x}$$

โดยที่ $i = \sqrt{-1}$ ตัวอย่างการใช้งานฟังก์ชันนี้ เช่น

```
-->sqrt([2, 4, -1, -4])                    //หาค่ารากที่สองของแต่ละสมาชิกในเวกเตอร์
 ans  =
    1.4142136    2.    i      2.i
```

สำหรับการหาค่ารากที่สองของจำนวนเชิงซ้อน ให้แทนเลขจำนวนเชิงซ้อนด้วยค่าเชิงขั้ว (polar representation) กล่าวคือถ้าตัวแปร x = a + bi เป็นค่าจำนวนเชิงซ้อน โดยที่ a และ b เป็นเลขจำนวนจริง และ i = $\sqrt{-1}$ ดังนั้นค่าเชิงขั้วของตัวแปร x คือ

$$x = re^{i\theta}$$

โดยที่ $r = \sqrt{a^2 + b^2}$ เป็นขนาด (magnitude) และ $\theta = \tan^{-1}\left(\frac{b}{a}\right)$ เป็นมุมเฟส (phase) ของเลขจำนวนเชิงซ้อน จากกฎของออยเลอร์ (Euler's formula) จะได้ว่า

$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

ฉะนั้นตัวแปร x สามารถเขียนให้อยู่ในรูปใหม่ได้ ดังนี้

$$x = r\{\cos(\theta) + i\sin(\theta)\} = a + bi$$

ซึ่งแสดงว่า $a = r\cos(\theta)$ และ $b = r\sin(\theta)$ ดังนั้นรากที่สองของตัวแปร x สามารถหาได้จาก

$$\sqrt{x} = \left(re^{i\theta}\right)^{1/2} = r^{1/2}e^{i\theta/2} = \sqrt{r}\cos\left(\frac{\theta}{2}\right) + i\sqrt{r}\sin\left(\frac{\theta}{2}\right)$$

ตัวอย่างการคำนวณ เช่น

```
-->sqrt([4, -4, 3+4*%i, 4*%i])
 ans  =
    2.    2.i    2. + i      1.4142136 + 1.4142136i
-->sqrt([9, -9; -2*%i, 1-2*%i])
 ans  =
    3.          3.i
    1. - i      1.2720196 - 0.7861514i
```

4.1.3 ฟังก์ชัน int(x)

เป็นฟังก์ชันที่ใช้ในการหาส่วนที่เป็นจำนวนเต็ม (integer part) ของตัวแปร x ตัวอย่างเช่น

```
-->int([0.1, 2.9, -1.5, 2.3*%i])
 ans  =
    0    2.  - 1.    2.i
```

สังเกตจะพบว่าผลลัพธ์ที่ได้ยังมีจุดทศนิยมตามหลังผลลัพธ์แต่ละตัว ซึ่งหมายความว่าผลลัพธ์ที่ได้เป็นส่วนที่เป็นจำนวนเต็มของตัวเลขแต่ละตัวเท่านั้น แต่ยังคงเป็นค่าจำนวนจริงที่มีความแม่นยำเป็นสองเท่า

4.1.4 ฟังก์ชัน modulo(m,n)

เป็นฟังก์ชันที่ใช้ในการหาเศษ (fraction) ที่ได้จากการหารค่าของตัวแปร m ด้วยค่าของตัวแปร n กล่าวคือ ถ้ากำหนดให้

$$\frac{m}{n} = q + \frac{r}{n}$$

โดยที่ q คือผลลัพธ์ที่ได้จากการหาร และ r คือเศษที่ได้จากการหาร ดังนั้นจะได้ว่า

```
r = modulo(m,n)
```

หรือสามารถหาค่าเศษที่ได้จากการหารนี้จากความสัมพันธ์

```
r = m - n.*int(m./n)
```

ผลลัพธ์ที่ได้สามารถเป็นได้ทั้งจำนวนเต็มบวก, จำนวนเต็มลบ, หรือจำนวนเต็มศูนย์ ทั้งนี้ขึ้นอยู่กับค่าของพารามิเตอร์ m และ n ตัวอย่างเช่น

```
-->m = [-3 -2 -1 0 1 2 3];
-->n = [2 2 2 2 2 2 2];
-->modulo(m,n)
```

```
 ans  =
  - 1.    0.  - 1.    0.    1.    0.    1.

-->r = m - n.*int(m./n)            //ได้ผลลัพธ์เท่ากับคำสั่ง modulo(m,n)
 r  =
  - 1.    0.  - 1.    0.    1.    0.    1.
```

4.1.5 ฟังก์ชัน ceil(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าจำนวนเต็มที่มีค่าใกล้กับค่า x ไปทางค่า $+\infty$ มากที่สุด ตัวอย่างเช่น

```
-->ceil([1.9  -2.5  2.1*%i  -3.8*%i  -%inf])
 ans  =
    2.  - 2.    3.i   - 3.i   -Inf
```

4.1.6 ฟังก์ชัน floor(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าจำนวนเต็มที่มีค่าใกล้กับค่า x ไปทางค่า $-\infty$ มากที่สุด ตัวอย่างเช่น

```
-->floor([1.9 -2.5  2.1*%i  -3.8*%i  %inf])
 ans  =
    1.  - 3.    2.i   - 4.i    Inf
```

4.1.7 ฟังก์ชัน round(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าจำนวนเต็มที่มีค่าใกล้กับค่า x มากที่สุด ตัวอย่างเช่น

```
-->round([1.9 -2.5  2.5  2.1*%i  -3.8*%i  %inf])
 ans  =
    2.  - 3.    3.    2.i  - 4.i   Inf
```

4.1.8 ฟังก์ชัน fix(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าจำนวนเต็มที่มีค่าใกล้กับค่า x ไปทางค่า 0 มากที่สุด นั่นคือมีค่าเท่ากับ

```
fix(x) = sign(x).*floor(abs(x))
```

ตัวอย่างเช่น

```
-->x = [-2.9  -1.1  1.5  3.8*%i  -5.2*%i]
 x  =
  - 2.9  - 1.1    1.5    3.8i  - 5.2i

-->fix(x)
 ans  =
  - 2.  - 1.    1.    3.i  - 5.i

-->sign(x).*floor(abs(x))
 ans  =
  - 2.  - 1.    1.    3.i  - 5.i
```

4.1.9 ฟังก์ชัน `rat(x)`

เป็นฟังก์ชันที่ใช้ในการประมาณค่าจำนวนจริงให้อยู่ในรูปของเลขเศษส่วน มีลักษณะการเรียกใช้งานดังนี้

```
[num, den] = rat(x, [tolerance])
```

โดยที่พารามิเตอร์

- `x` คือ เลขจำนวนจริงใดๆ
- `tolerance` คือ ค่าความคลาดเคลื่อนที่ยอมรับได้ โดยที่ค่าโดยปริยายมีค่าเท่ากับ `1.e-6*norm(x,1)`
- `num` และ `den` คือ เลขจำนวนเต็มซึ่งเป็นผลลัพธ์ของฟังก์ชันนี้ เมื่อ `num./den` จะมีค่าใกล้เคียงกับค่า `x` มากที่สุดในลักษณะที่ว่า `abs(num./den-x) <= tolerance*abs(x)`

ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->[num, den] = rat(0.333333333)          //นั่นคือ 1/3 = 0.3333333...
 den  =
    3.
 num  =
    1.

-->[num, den] = rat(%pi)
 den  =
    113.
```

```
 num  =
    355.

-->[355/113-%pi,  22/7-%pi]      //ตรวจสอบผลลัพธ์ที่ได้ว่ามีค่าใกล้เคียงกับค่า π หรือไม่
 ans  =                          //จะพบว่า 355/113 มีค่าใกล้เคียงค่า π มากกว่าค่า 22/7
    0.0000003    0.0012645
-->[num, den] = rat(%pi, 1.D-12)
 den  =
    364913.
 num  =
    1146408.

-->num/den - %pi                 //ค่าความคลาดเคลื่อนเท่ากับที่กำหนดไว้
 ans  =
    1.611D-12
```

4.1.10 ฟังก์ชัน sign(x)

ถ้าตัวแปร x เป็นค่าจำนวนจริง ฟังก์ชัน sign(x) จะมีค่าดังนี้

$$\text{sign}(x)=\begin{cases}+1 & ,x>0\\ 0 & ,x=0\\ -1 & ,x<0\end{cases}$$

กล่าวคือฟังก์ชันนี้จะให้ผลลัพธ์เป็นค่า +1 ถ้าตัวแปร x มีค่ามากกว่าศูนย์, ให้ผลลัพธ์เป็นค่า 0 ถ้าตัวแปร x มีค่าเท่ากับศูนย์, และให้ผลลัพธ์เป็นค่า -1 ถ้าตัวแปร x มีค่าน้อยกว่าศูนย์ ตัวอย่างเช่น

```
-->sign([-%inf  -0.01  0  0.02  %inf])
 ans  =
  - 1.  - 1.    0.    1.    1.
```

ในกรณีที่ตัวแปร x เป็นค่าจำนวนเชิงซ้อน ฟังก์ชันนี้จะให้ผลลัพธ์เป็น sign(x) = x./abs(x) ตัวอย่างเช่น

```
-->sign([%i  -%i  3+4*%i  3-4*%i])
 ans  =
    i    - i      0.6 + 0.8i    0.6 - 0.8i
```

4.1.11 ฟังก์ชัน roots(x)

เป็นฟังก์ชันที่ใช้ในการหารากหรือคำตอบของสมการพหุนาม โปรแกรม SCILAB สามารถรองรับพหุนามที่มีดีกรีน้อยกว่าหรือเท่ากับ 100 เท่านั้น ตัวอย่างเช่น

```
-->y = poly([1 3 5], 'x', 'r')     //สร้างสมการพหุนามจากคำตอบของสมการ
 y  =
                  2    3
  - 15 + 23x - 9x + x
-->roots(y)
 ans  =
    1.
    3.
    5.
```

4.1.12 ฟังก์ชัน real(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าจำนวนจริง (real number) ของตัวแปร x ตัวอย่างเช่น

```
-->real([0.1, %i, -1.5+2*%i, 2-%i])
 ans  =
    0.1    0.   - 1.5    2.
```

4.1.13 ฟังก์ชัน imag(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าจำนวนจินตภาพ (imaginary number) ของตัวแปร x ตัวอย่างเช่น

```
-->imag([0.1, %i, -1.5+2*%i, 2-%i])
 ans  =
    0.    1.    2.   - 1.
```

4.1.14 ฟังก์ชัน conj(x)

เป็นฟังก์ชันที่ใช้ในการทำสังยุคของจำนวนเชิงซ้อน (complex conjugate) ของตัวแปร x ตัวอย่างเช่น

```
-->conj([0.1, %i, -1.5+2*%i, 2-%i])
 ans  =
    0.1   - i    - 1.5 - 2.i    2. + i
```

ตารางที่ 4.2 ตัวอย่างฟังก์ชันเลขชี้กำลังและลอการิทึม

ฟังก์ชัน	คำอธิบาย
`exp(x)`	หาค่า e^x ของตัวแปร `x`
`log(x)`	หาค่า `log` ฐาน `e` ของตัวแปร `x`
`log2(x)`	หาค่า `log` ฐาน 2 ของตัวแปร `x`
`log10(x)`	หาค่า `log` ฐาน 10 ของตัวแปร `x`

4.2 ฟังก์ชันเลขชี้กำลังและลอการิทึม

ในส่วนนี้จะกล่าวถึงฟังก์ชันเลขชี้กำลังและลอการิทึม (exponential and logarithm functions) ตามที่แสดงในตารางที่ 4.2 ซึ่งมีลักษณะการใช้งานดังต่อไปนี้

4.2.1 ฟังก์ชัน `exp(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่า e^x (`e = 2.7182818`) ตัวอย่างเช่น

```
-->[exp(1) %e]
 ans  =
    2.7182818    2.7182818

-->x = log([1, %e, 10,  20]);

-->exp(x)
 ans  =
    1.    2.7182818    10.    20.
```

สำหรับการหาค่าฟังก์ชันเลขชี้กำลังของเลขจำนวนเชิงซ้อน `x = a + bi` โดยที่ `a` และ `b` เป็นเลขจำนวนจริง และ `i` = $\sqrt{-1}$ สามารถทำได้ดังนี้

$$\texttt{exp(x)} = e^{a+bi} = e^a e^{bi} = e^a\{\cos(b) + i\sin(b)\} = e^a\cos(b) + ie^a\sin(b)$$

ตัวอย่างเช่น

```
-->exp([1-%i  3+4*%i])
 ans  =
    1.4686939 - 2.2873553i  - 13.128783 - 15.200784i
```

4.2.2 ฟังก์ชัน log(x)

ค่าอินเวอร์ส (inverse) ของฟังก์ชันเลขชี้กำลังก็คือ ฟังก์ชันลอการิทึมธรรมชาติ (natural logarithm) หรือค่า log ฐาน e กล่าวคือถ้า y = log(x) ดังนั้น x = exp(y) = e^y ตัวอย่างเช่น

```
-->log([1, %e, 10, 20, 100])
 ans  =
    0.    1.    2.3025851    2.9957323    4.6051702
```

เช่นเดียวกันการหาค่าฟังก์ชันลอการิทึมธรรมชาติของเลขจำนวนเชิงซ้อน x = a + bi สามารถทำได้ดังนี้

$$\log(x) = \log\left(re^{i\theta}\right) = \log(r) + \log\left(e^{i\theta}\right) = \log(r) + i\theta$$

โดยที่ $r = \sqrt{a^2 + b^2}$ เป็นขนาด และ $\theta = \tan^{-1}\left(\frac{b}{a}\right)$ เป็นมุมของ x ตัวอย่างเช่น

```
-->log([1+2*%i,  3-4*%i])
 ans  =
    0.8047190 + 1.1071487i    1.6094379 - 0.9272952i
```

4.2.3 ฟังก์ชัน log2(x)

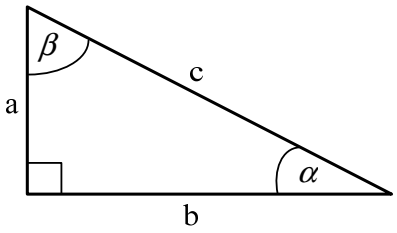
เป็นฟังก์ชันที่ใช้ในการหาค่า log ฐาน 2 ของค่า x ตัวอย่างเช่น

```
-->log2([1, 2, 4, 8])
 ans  =
    0.    1.    2.    3.
```

4.2.4 ฟังก์ชัน log10(x)

เป็นฟังก์ชันที่ใช้ในการหาค่า log ฐาน 10 ของค่า x ตัวอย่างเช่น

```
-->log10([1, 2, 10, 20, 100])
 ans  =
    0.    0.30103    1.    1.30103    2.
```



รูปที่ 4.1 สามเหลี่ยมมุมฉาก

สำหรับฟังก์ชันลอการิทึมที่มีฐานอื่นๆ ก็สามารถหาค่าได้โดยอาศัยความสัมพันธ์ดังนี้

$$\log_x(y) = \frac{\log_a(y)}{\log_a(x)}$$

นั่นคือ log ของจำนวน y ที่มีฐาน x มีค่าเท่ากับ log ของจำนวน y ที่มีฐาน a หารด้วย log ของจำนวน x ที่มีฐาน a โดยที่ a เป็นเลขจำนวนเต็มบวกใดๆ ตัวอย่างเช่น ถ้าต้องการหาค่า log ของ 81 ที่มีฐาน 3 ก็สามารถทำได้ดังนี้

```
-->y = [log(81)/log(3), log2(81)/log2(3), log10(81)/log10(3)]
 y  =
    4.    4.    4.
```

ซึ่งให้ผลลัพธ์เท่ากับ $\log_3(81) = \log_3(3^4) = 4\times\log_3(3) = 4\times1 = 4$

4.3 ฟังก์ชันตรีโกณมิติ

ฟังก์ชันตรีโกณมิติ (trigonometric function) ประกอบไปด้วย 6 ฟังก์ชัน ได้แก่ sine, cosine, tangent, secant, cosecant, และ cotangent โดยทั่วไปแล้วฟังก์ชันตรีโกณมิติเหล่านี้สามารถที่จะถูกกำหนดในรูปของมุมและความยาวของด้านต่างๆ ของรูปสามเหลี่ยมมุมฉากตามที่แสดงในรูปที่ 4.1 โดยที่ c จะถูกเรียกว่าด้านตรงข้ามมุมฉาก (hypotenuse) ถ้าพิจารณาที่มุม α (alpha) ด้าน b จะถูกเรียกว่าด้านประชิดมุม และด้าน a จะถูกเรียกว่าด้านตรงข้ามมุม จากรูปที่ 4.1 จะได้ความสัมพันธ์ของฟังก์ชันตรีโกณมิติต่างๆ ดังนี้

$$\sin(\alpha)=\frac{a}{c} \qquad \operatorname{cosec}(\alpha)=\frac{1}{\sin(\alpha)}=\frac{c}{a}$$

$$\cos(\alpha)=\frac{b}{c} \qquad \sec(\alpha)=\frac{1}{\cos(\alpha)}=\frac{c}{b}$$

$$\tan(\alpha)=\frac{\sin(\alpha)}{\cos(\alpha)}=\frac{a}{b} \qquad \cot(\alpha)=\frac{1}{\tan(\alpha)}=\frac{b}{a}$$

ในการใช้งานฟังก์ชันตรีโกณมิติ ค่ามุมที่ใช้หรือที่ได้รับจากฟังก์ชันทางตรีโกณมิติจะต้องมีหน่วยเป็นเรเดียน[21] (radian) ในกรณีที่ต้องการใช้ค่ามุมเป็นหน่วยองศา (degree) ก็สามารถทำได้โดยใช้ความสัมพันธ์ที่ว่า $\pi = 180^{\circ}$ ในทางกลับกันถ้าต้องการทราบว่าค่ามุมเท่าใดที่ทำให้ได้ผลลัพธ์เป็นค่าของฟังก์ชันตรีโกณมิตินั้นก็สามารถทำได้โดยใช้ฟังก์ชันตรีโกณมิติผกผัน (inverse trigonometric function) ซึ่งประกอบไปด้วย 6 ฟังก์ชัน ได้แก่ arcsine, arccosine, arctangent, arcsecant, arccosecant, และ arccotangent โดยมีความสัมพันธ์กับฟังก์ชันตรีโกณมิติดังต่อไปนี้

ถ้า $y=\sin(\alpha)$ จะได้ว่า $\alpha=\sin^{-1}(y)=\text{arcsine}(y)$

ถ้า $y=\cos(\alpha)$ จะได้ว่า $\alpha=\cos^{-1}(y)=\text{arccosine}(y)$

ถ้า $y=\tan(\alpha)$ จะได้ว่า $\alpha=\tan^{-1}(y)=\text{arctangent}(y)$

โปรแกรม SCILAB ได้เตรียมคำสั่งพื้นฐานสำหรับฟังก์ชันตรีโกณมิติและฟังก์ชันตรีโกณมิติผกผันไว้ตามที่แสดงในตารางที่ 4.3 ซึ่งแต่ละฟังก์ชันมีลักษณะการใช้งานดังต่อไปนี้

4.3.1 ฟังก์ชัน `sin(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่า sine ของค่า x โดยที่ x คือมุมที่มีหน่วยเป็นเรเดียน ตัวอย่างเช่น

```
-->y = sin([0, 1, %pi/2, %pi])
 y  =
    0.    0.8414710    1.    1.225D-16
```

[21] ค่าอัตราส่วนระหว่างความยาวส่วนโค้งของวงกลมต่อความยาวของรัศมีของวงกลม

ตารางที่ 4.3 ฟังก์ชันตรีโกณมิติและฟังก์ชันตรีโกณมิติผกผัน

ฟังก์ชัน	คำอธิบาย
`sin(x)`	หาค่า sine ของตัวแปร `x`
`cos(x)`	หาค่า cosine ของตัวแปร `x`
`tan(x)`	หาค่า tangent ของตัวแปร `x`
`asin(y)`	หาค่า sine inverse ของตัวแปร `y`
`acos(y)`	หาค่า cosine inverse ของตัวแปร `y`
`atan(y)`	หาค่า tangent inverse ของตัวแปร `y`

สังเกตจะพบว่าค่าสุดท้ายมีค่าน้อยมากหรือสามารถประมาณได้เท่ากับค่าศูนย์ (ใช้คำสั่ง `clean` ช่วยในการปัดค่าได้) ซึ่งจะทำให้สอดคล้องกับหลักความจริงที่ว่า `sin(180°) = 0` ตามที่ต้องการ

4.3.2 ฟังก์ชัน `cos(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่า cosine ของค่า `x` โดยที่ `x` คือมุมที่มีหน่วยเป็นเรเดียน ตัวอย่างเช่น

```
-->y = clean(cos([0, 1, %pi/2, %pi]))
 y  =
    1.    0.5403023    0.   - 1.
```

4.3.3 ฟังก์ชัน `tan(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่า tangent ของค่า `x` โดยที่ `x` คือมุมที่มีหน่วยเป็นเรเดียน ตัวอย่างเช่น

```
-->y = tan([0, 1, %pi/2, %pi])
 y  =
    0.    1.5574077    1.633D+16   - 1.225D-16
```

4.3.4 ฟังก์ชัน `asin(y)`

เป็นฟังก์ชันที่ใช้ในการหาค่า arcsine(y) นั่นคือถ้า `y = sin(x)` จะได้ว่า `x = asin(y)` โดยที่ `y` จะมีค่าอยู่ระหว่างค่า `-1` ถึง `1` และ `x` คือผลลัพธ์ที่ได้ซึ่งจะเป็นมุมที่มีค่าอยู่ระหว่าง $-\pi/2$ ถึง $\pi/2$ เรเดียน ตัวอย่างเช่น

```
-->y = sin([0, 1, %pi/2, -%pi/2]);
-->x = asin(y)
 ans  =
    0.    1.    1.5707963   - 1.5707963
```

4.3.5 ฟังก์ชัน acos(y)

เป็นฟังก์ชันที่ใช้ในการหาค่า arccosine(y) นั่นคือถ้า y = cos(x) จะได้ว่า x = acos(y) โดยที่ y จะมีค่าอยู่ระหว่างค่า -1 ถึง 1 และ x คือผลลัพธ์ที่ได้ซึ่งจะเป็นมุมที่มีค่าอยู่ระหว่าง 0 ถึง π เรเดียน ตัวอย่างเช่น

```
-->y = cos([0,  1,  %pi/2,  %pi]);
-->x = acos(y)
 ans  =
    0.    1.    1.5707963    3.1415927
```

4.3.6 ฟังก์ชัน atan(y)

เป็นฟังก์ชันที่ใช้ในการหาค่า arctangent(y) นั่นคือถ้า y = tan(x) จะได้ว่า x = atan(y) โดยที่ y เป็นจำนวนจริงใดๆ และ x คือผลลัพธ์ที่ได้ซึ่งจะเป็นมุมที่มีค่าอยู่ระหว่าง $-\pi/2$ ถึง $\pi/2$ เรเดียน ตัวอย่างเช่น

```
-->y = tan([0, 1, %pi/2, -%pi/2]);
-->x = atan(y)
 ans  =
    0.    1.    1.5707963   - 1.5707963
```

นอกจากนี้คำสั่ง atan ยังสามารถเรียกใช้งานในอีกลักษณะหนึ่งได้คือ atan(y, x) ซึ่งใช้ในการหาค่า arctangent (หรือค่าอินเวอร์สของ tangent) ของค่า y หารด้วยค่า x ซึ่งผลลัพธ์ที่ได้จะเป็นมุมที่มีค่าอยู่ระหว่าง $-\pi$ ถึง π เรเดียน โดยจะขึ้นอยู่กับเครื่องหมายของตัวแปร y และ x เช่น

```
-->x = [1, %i, -1, -%i];
-->phase = atan(imag(x), real(x))
 phase  =
    0.    1.5707963    3.1415927   - 1.5707963
```

สำหรับค่าตรีโกณมิติอื่นๆ ได้แก่ secant, cosecant, cotangent, arcsecant, arccosecant, และ arccotangent สามารถคำนวณหาได้โดยการดัดแปลงจากฟังก์ชันที่โปรแกรม SCILAB มีอยู่แล้ว ตามที่อธิบายไว้ข้างต้น

4.4 ฟังก์ชันไฮเพอร์โบลิก

ฟังก์ชันไฮเพอร์โบลิก (hyperbolic function) เป็นฟังก์ชันที่เป็นผลมาจากการรวมกันของฟังก์ชันลอการิทึมธรรมชาติ e^x ประกอบด้วย 6 ฟังก์ชัน คือ hyperbolic sine (sinh), hyperbolic cosine (cosh), hyperbolic tangent (tanh), hyperbolic secant (sech), hyperbolic cosecant (csch), และ hyperbolic cotangent (coth) ซึ่งมีรูปแบบดังนี้

$$\sinh(x) = \frac{e^x - e^{-x}}{2} \qquad \operatorname{csch}(x) = \frac{1}{\sinh(x)}$$

$$\cosh(x) = \frac{e^x + e^{-x}}{2} \qquad \operatorname{sech}(x) = \frac{1}{\cosh(x)}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad \coth(x) = \frac{1}{\tanh(x)}$$

ในทำนองเดียวกันค่าอินเวอร์สของฟังก์ชันไฮเพอร์โบลิก (หรือเรียกว่าฟังก์ชันไฮเพอร์โบลิกผกผัน) เหล่านี้ก็สามารถหาได้โดยใช้ความสัมพันธ์ดังนี้

ถ้า $y = \sinh(x)$ จะได้ว่า $x = \sinh^{-1}(y) = \operatorname{asinh}(y)$

ถ้า $y = \cosh(x)$ จะได้ว่า $x = \cosh^{-1}(y) = \operatorname{acosh}(y)$

ถ้า $y = \tanh(x)$ จะได้ว่า $x = \tanh^{-1}(y) = \operatorname{atanh}(y)$

โปรแกรม SCILAB ได้เตรียมคำสั่งพื้นฐานสำหรับฟังก์ชันไฮเพอร์โบลิกและฟังก์ชันไฮเพอร์โบลิกผกผันไว้ตามที่แสดงในตารางที่ 4.4 ซึ่งแต่ละฟังก์ชันมีลักษณะการใช้งานดังต่อไปนี้

ตารางที่ 4.4 ฟังก์ชันไฮเพอร์โบลิกและฟังก์ชันไฮเพอร์โบลิกผกผัน

ฟังก์ชัน	คำอธิบาย
`sinh(x)`	ใช้หาค่า hyperbolic sine ของตัวแปร `x`
`cosh(x)`	ใช้หาค่า hyperbolic cosine ของตัวแปร `x`
`tanh(x)`	ใช้หาค่า hyperbolic tangent ของตัวแปร `x`
`asinh(y)`	ใช้หาค่า hyperbolic sine inverse ของตัวแปร `y`
`acosh(y)`	ใช้หาค่า hyperbolic cosine inverse ของตัวแปร `y`
`atanh(y)`	ใช้หาค่า hyperbolic tangent inverse ของตัวแปร `y`

4.4.1 ฟังก์ชัน `sinh(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่า hyperbolic sine ของค่า `x` ตัวอย่างเช่น

```
-->y = sinh([0, 1, -1])
 y  =
    0.    1.1752012  - 1.1752012
```

4.4.2 ฟังก์ชัน `cosh(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่า hyperbolic cosine ของค่า `x` ตัวอย่างเช่น

```
-->y = cosh([0, 1, -1])
 y  =
    1.    1.5430806    1.5430806
```

4.4.3 ฟังก์ชัน `tanh(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่า hyperbolic tangent ของค่า `x` ตัวอย่างเช่น

```
-->y = tanh([0, 1, -1])
 y  =
    0.    0.7615942  - 0.7615942
```

4.4.4 ฟังก์ชัน asinh(y)

เป็นฟังก์ชันที่ใช้ในการหาค่าอินเวอร์สของ hyperbolic sine ของ y ซึ่งมีค่าเท่ากับ

$$\text{asinh}(y) = \ln\left(y + \sqrt{y^2 + 1}\right)$$

เมื่อ ln(x) คือฟังก์ชันลอการิทึมธรรมชาติของจำนวน x ตัวอย่างเช่น

```
-->y = sinh([0, 1, -1]);
-->x = asinh(y)
 x  =
    0.    1.  - 1.
-->z = log(y + sqrt(y.^2 + 1))     //มีค่าเท่ากับการใช้คำสั่ง asinh
 z  =
    0.    1.  - 1.
```

4.4.5 ฟังก์ชัน acosh(y)

เป็นฟังก์ชันที่ใช้ในการหาค่าอินเวอร์สของ hyperbolic cosine ของค่า y ซึ่งมีค่าเท่ากับ

$$\text{acosh}(y) = \ln\left(y + \sqrt{y^2 - 1}\right)$$

โดยที่ค่า $y \geq 1$ และผลลัพธ์ที่ได้จะเป็นค่าสัมบูรณ์เสมอ ตัวอย่างเช่น

```
-->y = cosh([0, 1, -1]);
-->x = acosh(y)
 x  =
    0.    1.    1.                  //ผลลัพธ์ที่ได้จะเป็นค่าสัมบูรณ์
-->z = log(y + sqrt(y.^2 - 1))     //มีค่าเท่ากับการใช้คำสั่ง acosh
 z  =
    0.    1.    1.
```

4.4.6 ฟังก์ชัน atanh(y)

เป็นฟังก์ชันที่ใช้ในการหาค่าอินเวอร์สของ hyperbolic tangent ของค่า y ซึ่งมีค่าเท่ากับ

$$\text{atanh}(y) = \frac{1}{2}\ln\left(\frac{1+y}{1-y}\right) \quad \text{สำหรับ } |y| < 1$$

ตัวอย่างเช่น

```
-->y = tanh([0, 1, -1]);
-->x = atanh(y)
 x  =
    0.    1.   - 1.
-->z = 0.5*log((1+y)./(1-y))      //มีค่าเท่ากับการใช้คำสั่ง atanh
 z  =
    0.    1.   - 1.
```

สำหรับค่าของฟังก์ชันไฮเพอร์โบลิกและฟังก์ชันไฮเพอร์โบลิกผกผันอื่นๆ ก็สามารถที่จะคำนวณหาได้โดยการดัดแปลงจากฟังก์ชันที่โปรแกรม SCILAB มีอยู่แล้ว ตามที่อธิบายไว้ข้างต้น

4.5 ฟังก์ชันพื้นฐานทางสถิติ

โปรแกรม SCILAB ได้เตรียมฟังก์ชันจำนวนมากสำหรับใช้งานทางด้านสถิติ ตารางที่ 4.5 แสดงตัวอย่างฟังก์ชันพื้นฐานสำหรับใช้งานทางด้านสถิติ ซึ่งแต่ละฟังก์ชันมีลักษณะการใช้งานดังนี้

4.5.1 ฟังก์ชัน min(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าต่ำสุด (minimum) ของตัวเลขทั้งหมดภายในตัวแปร x ดังแสดงในตัวอย่างต่อไปนี้

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2]
 x  =
    1.    2.    3.    4.
    0.    1.    5.    3.
    2.    6.    1.    2.
```

ตารางที่ 4.5 ตัวอย่างฟังก์ชันพื้นฐานทางสถิติ

ฟังก์ชัน	คำอธิบาย
`min(x)`	หาค่าต่ำสุด (minimum) ของตัวเลขทั้งหมดในตัวแปร `x`
`max(x)`	หาค่าสูงสุด (maximum) ของตัวเลขทั้งหมดในตัวแปร `x`
`mean(x)`	หาค่าเฉลี่ย (mean) ของตัวเลขทั้งหมดในตัวแปร `x` (ถือเป็นค่าเฉลี่ยเลขคณิต)
`median(x)`	หาค่ามัธยฐาน (median) ของตัวเลขทั้งหมดในตัวแปร `x`
`stdev(x)`	หาค่าเบี่ยงเบนมาตรฐาน (standard deviation) ของตัวเลขทั้งหมดในตัวแปร `x`
`sum(x)`	หาค่าผลบวกของตัวเลขทั้งหมดในตัวแปร `x`
`cumsum(x)`	หาค่าผลบวกสะสม (cumulative sum) ของตัวเลขทั้งหมดในตัวแปร `x`
`prod(x)`	หาค่าผลคูณของตัวเลขทั้งหมดในตัวแปร `x`
`cumprod(x)`	หาค่าผลคูณสะสม (cumulative product) ของตัวเลขทั้งหมดในตัวแปร `x`
`sort(x)`	เรียงลำดับตัวเลขทั้งหมดในตัวแปร `x` จากค่ามากไปหาค่าน้อย
`histplot(n,x)`	วาดรูปฮิสโตแกรม (histogram) ของค่าทั้งหมดในเวกเตอร์ `x` เป็นจำนวน `n` ช่วงระหว่างค่าต่ำสุดและค่าสูงสุดของเวกเตอร์ `x`
`variance(x)`	หาค่าความแปรปรวน (variance) ของตัวเลขทั้งหมดในตัวแปร `x`
`geomean(x)`	หาค่าเฉลี่ยเรขา (geometric mean) ของตัวเลขทั้งหมดในตัวแปร `x`
`harmean(x)`	หาค่าเฉลี่ยฮาร์มอนิก (harmonic mean) ของตัวเลขทั้งหมดในตัวแปร `x`
`nfreq(x)`	หาความถี่ (frequency) ของตัวเลขทั้งหมดในตัวแปร `x`

```
-->min(x)                          //หาค่าต่ำสุดของตัวเลขทั้งหมดในเมทริกซ์ x
 ans  =
    0.

-->min(x, 'r')                     //หาค่าต่ำสุดของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    0.    1.    1.    2.

-->min(x, 'c')                     //หาค่าต่ำสุดของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    1.
    0.
    1.
```

นอกจากนี้ฟังก์ชัน `min` ยังมีลักษณะการใช้งานแบบอื่นอีก ลองศึกษาได้จากคำสั่ง `help`

4.5.2 ฟังก์ชัน max(x)

ฟังก์ชันนี้มีรูปแบบการใช้เหมือนกับฟังก์ชัน min ทุกประการ เพียงแต่จะให้ค่าสูงสุด (maximum) แทนค่าต่ำสุด ตัวอย่างเช่น

```
--> x = [1 2 3 4; 0 1 5 3; 2 6 1 2];

-->max(x)                    //หาค่าสูงสุดของตัวเลขทั้งหมดในเมทริกซ์ x
 ans  =
    6.

-->max(x, 'r')               \//หาค่าสูงสุดของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    2.    6.    5.    4.

-->max(x, 'c')               //หาค่าสูงสุดของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    4.
    5.
    6.
```

4.5.3 ฟังก์ชัน mean(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าเฉลี่ยเลขคณิต (arithmetic mean) ของตัวเลขทั้งหมดในตัวแปร x สมมติว่า x = [x_1 x_2 ... x_N] มีสมาชิกทั้งหมด N ตัว ดังนั้นค่าเฉลี่ยเลขคณิต am หาได้จาก

$$am = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{x_1 + x_2 + \ldots + x_N}{N}$$

ตัวอย่างเช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];

-->mean(x)                   //หาค่าเฉลี่ยของตัวเลขทั้งหมดในเมทริกซ์ x
 ans  =
    2.5

-->mean(x, 'r')              //หาค่าเฉลี่ยของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
```

```
 ans  =
    1.    3.    3.    3.
```

```
-->mean(x, 'c')              //หาค่าเฉลี่ยของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    2.5
    2.25
    2.75
```

4.5.4 ฟังก์ชัน median(x)

เป็นฟังก์ชันที่ใช้ในการหาค่ามัธยฐาน (median) ของตัวเลขทั้งหมดในตัวแปร x ตัวอย่างเช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];
```

```
-->median(x)                 //หาค่ามัธยฐานของตัวเลขทั้งหมดในเมทริกซ์ x
 ans  =
    2.
```

```
-->median(x, 'r')            //หาค่ามัธยฐานของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    1.    2.    3.    3.
```

```
-->median(x, 'c')            //หาค่ามัธยฐานของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    2.5
    2.
    2.
```

4.5.5 ฟังก์ชัน stdev(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าเบี่ยงเบนมาตรฐาน (standard deviation) ของตัวเลขทั้งหมดในตัวแปร x ตัวอย่างเช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];
```

```
-->stdev(x)          //หาค่าเบี่ยงเบนมาตรฐานของตัวเลขทั้งหมดในเมทริกซ์ x
 ans  =
    1.7837652
```

```
-->stdev(x, 'r')      //หาค่าเบี่ยงเบนมาตรฐานของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    1.    2.6457513    2.    1.

-->stdev(x, 'c')      //หาค่าเบี่ยงเบนมาตรฐานของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    1.2909944
    2.2173558
    2.2173558
```

4.5.6 ฟังก์ชัน sum(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าผลบวกของตัวเลขทั้งหมดในตัวแปร x ตัวอย่างเช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];

-->sum(x)                    //หาค่าผลบวกของตัวเลขทั้งหมดในเมทริกซ์ x
 ans  =
    30.

-->sum(x, 'r')               //หาค่าผลบวกของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    3.    9.    9.    9.

-->sum(x, 'c')               //หาค่าผลบวกของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    10.
    9.
    11.
```

4.5.7 ฟังก์ชัน cumsum(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าผลบวกสะสม (cumulative sum) ของตัวเลขทั้งหมดในตัวแปร x ตัวอย่างเช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];

-->cumsum(x)          //หาค่าผลบวกสะสมของตัวเลขทั้งหมดในเมทริกซ์ x
```

```
 ans  =
    1.    5.    15.    25.
    1.    6.    20.    28.
    3.    12.   21.    30.
-->cumsum(x, 'r')          //หาค่าผลบวกสะสมของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    1.    2.    3.    4.
    1.    3.    8.    7.
    3.    9.    9.    9.
-->cumsum(x, 'c')          //หาค่าผลบวกสะสมของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    1.    3.    6.    10.
    0.    1.    6.    9.
    2.    8.    9.    11.
```

4.5.8 ฟังก์ชัน prod(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าผลคูณของตัวเลขทั้งหมดในตัวแปร x ตัวอย่างเช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];
-->prod(x)                  //หาค่าผลคูณตัวเลขทั้งหมดในเมทริกซ์ x
 ans  =
    0.
-->prod(x, 'r')             //หาค่าผลคูณของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    0.    12.   15.   24.
-->prod(x, 'c')             //หาค่าผลคูณของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    24.
    0.
    24.
```

4.5.9 ฟังก์ชัน cumprod(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าผลคูณสะสม (cumulative product) ของตัวเลขทั้งหมดในตัวแปร x ตัวอย่างเช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];

-->cumprod(x)                    //หาค่าผลคูณสะสมตัวเลขทั้งหมดในเมทริกซ์ x
 ans  =
    1.    0.    0.    0.
    0.    0.    0.    0.
    0.    0.    0.    0.

-->cumprod(x, 'r')               //หาค่าผลคูณสะสมของตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    1.    2.    3.    4.
    0.    2.    15.   12.
    0.    12.   15.   24.

-->cumprod(x, 'c')               //หาค่าผลคูณสะสมของตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
    1.    2.    6.    24.
    0.    0.    0.    0.
    2.    12.   12.   24.
```

4.5.10 ฟังก์ชัน sort(x)

เป็นฟังก์ชันที่ใช้ในการเรียงลำดับตัวเลขทั้งหมดในตัวแปร x จากค่ามากไปหาค่าน้อยตัวอย่างเช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];

-->sort(x)                       //เรียงลำดับตัวเลขทั้งหมดในเมทริกซ์ x  โดยผลลัพธ์ที่ได้
 ans  =                          //จะเรียงจากบนลงล่าง และจากซ้ายไปขวา
    6.    3.    2.    1.
    5.    3.    2.    1.
    4.    2.    1.    0.

-->sort(x, 'r')                  //เรียงลำดับตัวเลขทั้งหมดในแต่ละแถวของเมทริกซ์ x
 ans  =
    2.    6.    5.    4.
    1.    2.    3.    3.
    0.    1.    1.    2.

-->sort(x, 'c')                  //เรียงลำดับตัวเลขทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ x
 ans  =
```

```
    4.    3.    2.    1.
    5.    3.    1.    0.
    6.    2.    2.    1.
```

ถ้าต้องการเรียงลำดับข้อมูลแบบอื่นๆ เช่น เรียงลำดับจากค่าน้อยไปหาค่ามาก ก็สามารถทำได้โดยใช้คำสั่ง `gsort` ซึ่งมีลักษณะการใช้งานหลายรูปแบบ (ศึกษารายละเอียดในคำสั่ง `help`)

4.5.11 ฟังก์ชัน `histplot(n,x)`

เป็นฟังก์ชันที่ใช้ในการวาดรูปฮิสโตแกรม (histogram) ของค่าทั้งหมดในเวกเตอร์ `x` เป็นจำนวน `n` ช่วงระหว่างค่าต่ำสุดและค่าสูงสุดของเวกเตอร์ `x` ตัวอย่างเช่น

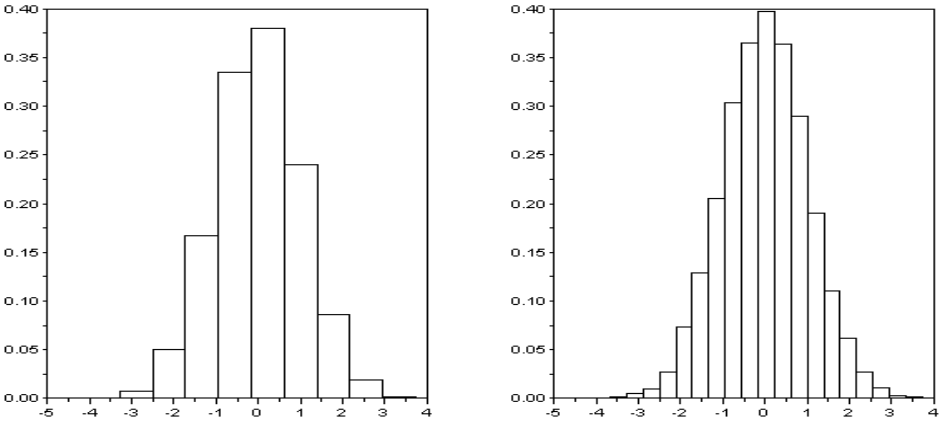
```
-->d = rand(1, 10000, 'normal');
-->subplot(1,2,1);  histplot(10, d);          //รูปที่ 4.1 ด้านซ้าย
-->subplot(1,2,2);  histplot(20, d);          //รูปที่ 4.1 ด้านขวา
```

คำสั่งแรกจะทำการสร้างจำนวนสุ่ม (random number) จำนวน 10000 ตัว (บรรจุไว้ในเวกเตอร์ขนาด 1×10000) โดยมีลักษณะการแจกแจงปรกติ (normal distribution) หรือการแจกแจงแบบเกาส์เซียน (Gaussian distribution) นั่นคือมีค่าเฉลี่ย (mean) เท่ากับค่า 0 และมีค่าความแปรปรวน (variance) เท่ากับค่า 1 จากนั้นก็ทำการวาดรูปฮิสโตแกรมของจำนวนสุ่มทั้งหมดโดยแบ่งข้อมูลเป็น 10 ช่วง (รูปที่ 4.1 ด้านซ้าย) และแบ่งข้อมูลเป็น 20 ช่วง (รูปที่ 4.1 ด้านขวา) ดูรายละเอียดของการสร้างจำนวนสุ่มโดยใช้คำสั่ง `rand` ในหัวข้อที่ 4.6.4

4.5.12 ฟังก์ชัน `variance(x)`

เป็นฟังก์ชันที่ใช้ในการหาค่าความแปรปรวน (variance) ของตัวเลขทั้งหมดในตัวแปร `x` เช่น

```
-->x = [1 2 3 4; 0 1 5 3; 2 6 1 2];
-->variance(x)
 ans  =
    3.1818182
```



รูปที่ 4.1 ตัวอย่างรูปฮิสโตแกรม

```
-->variance(x, 'r')
 ans  =
    1.    7.    4.    1.
-->variance(x, 'c')
 ans  =
    1.6666667
    4.9166667
    4.9166667
```

4.5.13 ฟังก์ชัน geomean(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าเฉลี่ยเรขาคณิต (geometric mean) ของตัวเลขทั้งหมดในตัวแปร x สมมติว่า x = [x_1 x_2 ... x_N] มีสมาชิกทั้งหมด N ตัว ดังนั้นค่าเฉลี่ยเรขาคณิต gm หาได้จาก

$$gm = \sqrt[N]{\sum_{i=1}^{N} x_i} = \sqrt[N]{x_1 \times x_2 \times \ldots \times x_N}$$

ตัวอย่างเช่น

```
-->x = [2 4 6];
-->gm = geomean(x)
```

```
 gm   =
    4.

-->gm = (2*4*8)^(1/3)                    //มีค่าเท่ากับการใช้คำสั่ง geomean(x)
 gm   =
    4.
```

4.5.14 ฟังก์ชัน harmean(x)

เป็นฟังก์ชันที่ใช้ในการหาค่าเฉลี่ยฮาร์มอนิก (harmonic mean) ของตัวเลขทั้งหมดในตัวแปร x สมมติว่า x = [x_1 x_2 ... x_N] มีสมาชิกทั้งหมด N ตัว ดังนั้นค่าเฉลี่ยฮาร์มอนิก hm หาได้จาก

$$hm = \frac{1}{\frac{1}{N}\sum_{i=1}^{N} x_i} = \frac{N}{\sum_{i=1}^{N} x_i}$$

ตัวอย่างเช่น

```
-->x = [2 4 8];
-->hm = harmean(x)
 hm   =
    3.4285714

-->hm = 3/(1/2 + 1/4 + 1/8)              //มีค่าเท่ากับการใช้คำสั่ง harmean(x)
 hm   =
    3.4285714
```

4.5.15 ฟังก์ชัน nfreq(x)

เป็นฟังก์ชันที่ใช้ในการหาความถี่ (frequency) ของตัวเลขทั้งหมดในตัวแปร x ตัวอย่างเช่น

```
-->x = [2 8 0 3 7 6 8 7 9 7]
 x   =
    2.    8.    0.    3.    7.    6.    8.    7.    9.    7.

-->m = nfreq(x)
 m   =
```

```
 2.    1.
 8.    2.                          //บอกว่ามีเลข 8 อยู่ 2 ตัว
 0.    1.
 3.    1.
 7.    3.                          //บอกว่ามีเลข 7 อยู่ 3 ตัว
 6.    1.
 9.    1.
```

ผลลัพธ์ที่ได้บอกให้ทราบว่าเวกเตอร์ x มีเลข 8 อยู่ 2 ตัว, เลข 7 อยู่ 3 ตัว, และตัวเลขที่เหลือมีอยู่อย่างละ 1 ตัว

นอกจากนี้โปรแกรม SCILAB ยังมีฟังก์ชันทางสถิติที่น่าสนใจอีกจำนวนมากในไลบรารี statslib ซึ่งสามารถเรียกดูได้ดังนี้

```
-->statslib
 statslib  =
 Functions files location :SCI\macros\statistics\
 center    correl    cmoment   covar     ftest     ftuneq    geomean
 harmean   iqr       mvvacov   meanf     mvcorrel  moment    mean
 median    msd       mad       nfreq     nanmeanf  nand2mean
 nansum    nanmean   nancumsum           nanstdev  nanmin
 nanmedian           nanmax    perctl    pca       quart     regress
 strange   samplef   sample    samwr     st_deviation        stdev
 stdevf    trimmean  thrownan  tabul     variance  variancef
 wcenter
```

ตัวอย่างฟังก์ชันที่น่าสนใจ เช่น

- perctl ทำหน้าที่คำนวณหาเปอร์เซ็นไทล์ (percentile)
- quart ทำหน้าที่คำนวณหาควอร์ไทล์ (quartile)
- correl ทำหน้าที่คำนวณหาค่าสหสัมพันธ์ (correlation)
- covar ทำหน้าที่คำนวณหาค่าความแปรปรวนร่วมเกี่ยว (covariance)
- moment ทำหน้าที่คำนวณหาค่าโมเมนต์ (moment)
- msd ทำหน้าที่คำนวณหาค่าเบี่ยงเบนกำลังสองเฉลี่ย (mean squared deviation)

สำหรับผู้สนใจสามารถศึกษารายละเอียดเพิ่มเติมของฟังก์ชันทางสถิติต่างๆ ได้จากคำสั่ง help

ตารางที่ 4.6 ตัวอย่างเมทริกซ์พิเศษในโปรแกรม SCILAB

คำสั่ง	คำอธิบาย
`eye`	เมทริกซ์เอกลักษณ์ (identity matrix)
`ones`	เมทริกซ์ค่าหนึ่ง (one matrix)
`zeros`	เมทริกซ์ค่าศูนย์ (zero matrix)
`rand`	เมทริกซ์สุ่ม (random matrix)
`diag`	เมทริกซ์ทแยงมุม (diagonal matrix)
`tril`	เมทริกซ์สามเหลี่ยมด้านล่าง (lower triangular matrix)
`triu`	เมทริกซ์สามเหลี่ยมด้านบน (upper triangular matrix)
`testmatrix`	เมทริกซ์รูปแบบพิเศษ
`toeplitz`	เมทริกซ์ Toeplitz (toeplitz matrix)
`spones`	เมทริกซ์มากเลขศูนย์ (sparse matrix)
`sylm`	เมทริกซ์ Sylvester (Sylvester matrix)

4.6 เมทริกซ์พิเศษ

ในการประยุกต์ใช้งานเมทริกซ์บางครั้งมีความจำเป็นต้องสร้างเมทริกซ์ที่มีค่าเฉพาะหรือมีรูปแบบที่เป็นมาตรฐาน เช่น ต้องการสร้างเมทริกซ์ที่มีค่าเป็นหนึ่งทั้งหมดขนาด `m`×`n` โดยที่ `m` และ `n` มีค่ามาก ถ้าสร้างเมทริกซ์นี้โดยการพิมพ์ค่าแต่ละค่าเข้าไปตามที่ได้อธิบายไว้ในหัวข้อที่ผ่านมา อาจจะทำให้เสียเวลามากและอาจเกิดข้อผิดพลาดได้ง่าย ดังนั้นโปรแกรม SCILAB จึงได้เตรียมฟังก์ชันพื้นฐานสำหรับสร้างเมทริกซ์พิเศษหลายรูปแบบขึ้นมาไว้ใช้งานตามตารางที่ 4.6 ในส่วนนี้จะอธิบายเฉพาะเมทริกซ์พิเศษที่ใช้บ่อยดังต่อไปนี้ (ผู้สนใจสามารถศึกษารายละเอียดลักษณะการใช้งานคำสั่งเหล่านี้เพิ่มเติมได้จากคำสั่ง `help`)

4.6.1 เมทริกซ์เอกลักษณ์

เมทริกซ์เอกลักษณ์ (identity matrix) คือเมทริกซ์ที่มีสมาชิกที่มีค่าเป็นค่า `1` ปรากฏอยู่เฉพาะในแนวเส้นทแยงมุมหลัก (main diagonal) ส่วนสมาชิกที่ตำแหน่งอื่นๆ จะมีค่าเป็นค่า `0` คุณสมบัติของเมทริกซ์เอกลักษณ์แบบจัตุรัส (เมทริกซ์ที่มีจำนวนแถวเท่ากับจำนวนแนวตั้ง) คือเมื่อคูณเมทริกซ์ใดด้วยเมทริกซ์เอกลักษณ์แบบจัตุรัสแล้ว ผลลัพธ์ที่ได้ก็คือเมทริกซ์เดิมนั้น ในโปรแกรม SCILAB การสร้างเมทริกซ์เอกลักษณ์ทำได้โดยใช้คำสั่ง `eye` เช่น

```
-->A = eye(3)                    //สร้างเมทริกซ์เอกลักษณ์ที่มีขนาดเท่ากับค่าสเกลาร์
    1.                           //นั่นคือเมทริกซ์ขนาด 1×1

-->A = eye(3,3)                  //สร้างเมทริกซ์เอกลักษณ์ที่มีขนาด 3×3
 A  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.

-->B = [1 2; 3 4];

-->C = eye(B)                    //สร้างเมทริกซ์เอกลักษณ์ขนาดเท่าเมทริกซ์ B
 C  =
    1.    0.
    0.    1.

-->B*eye(B)                      //ผลลัพธ์ที่ได้คือเมทริกซ์ B
 ans  =
    1.    2.
    3.    4.

-->A = eye(3,4)                  //เมทริกซ์เอกลักษณ์ไม่จำเป็นต้องเป็นเมทริกซ์จัตุรัสก็ได้
 A  =
    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
```

4.6.2 เมทริกซ์ค่าหนึ่ง

เมทริกซ์ค่าหนึ่ง (one matrix) เป็นเมทริกซ์ที่สมาชิกทุกตัวในเมทริกซ์มีค่าเป็นค่า 1 การสร้างเมทริกซ์ค่าหนึ่งในโปรแกรม SCILAB สามารถทำได้โดยใช้คำสั่ง ones ตัวอย่างเช่น

```
-->A = ones(2, 3)                //สร้างเมทริกซ์ค่าหนึ่งขนาด 2 แถวและ 3 แนวตั้ง
 A  =
    1.    1.    1.
    1.    1.    1.

-->A = ones(1:3)                 //สร้างเมทริกซ์ค่าหนึ่งขนาด 1 แถวและ 3 แนวตั้ง
 A  =
    1.    1.    1.
```

```
-->v = [1 2 3 4];
-->A = ones(v)                          //สร้างเมทริกซ์ค่าหนึ่งขนาดเท่ากับเวกเตอร์ v
 ans  =
    1.    1.    1.    1.
```

4.6.3 เมทริกซ์ค่าศูนย์

เมทริกซ์ค่าศูนย์ (zero matrix) เป็นเมทริกซ์ที่สมาชิกทุกตัวในเมทริกซ์มีค่าเป็นค่า 0 มีลักษณะการทำงานเหมือนกับเมทริกซ์ค่าหนึ่งทุกประการ การสร้างเมทริกซ์ค่าศูนย์ในโปรแกรม SCILAB สามารถทำได้โดยใช้คำสั่ง `zeros` เช่น

```
-->A = zeros(2, 3)
 A  =
    0.    0.    0.
    0.    0.    0.
```

4.6.4 เมทริกซ์สุ่ม

เมทริกซ์สุ่ม (random matrix) เป็นเมทริกซ์ที่มีสมาชิกเป็นจำนวนสุ่ม (random number) การสร้างเมทริกซ์สุ่มในโปรแกรม SCILAB สามารถทำได้โดยใช้คำสั่ง `rand` ซึ่งมีลักษณะการเรียกใช้งานหลายรูปแบบ ดังนี้

```
rand(m1, m2, ..., [key])
rand(X, [key])
```

โดยที่พารามิเตอร์

- `mi` คือ เลขจำนวนเต็มบวก (เมื่อ `i = 1,2,3,...`) ใช้กำหนดขนาดของเมทริกซ์สุ่มที่จะสร้างขึ้นมา เช่น `rand(m1, m2)` หมายถึงสร้างเมทริกซ์สุ่มขนาด `m1` แนวนอน และ `m2` แนวตั้ง
- `X` คือ เมทริกซ์ใดๆ (นั่นคือสั่งให้สร้างเมทริกซ์สุ่มขนาดเท่ากับเมทริกซ์ `X`)
- `key` เป็นตัวเลือกที่กำหนดลักษณะการแจกแจง (distribution) ของจำนวนสุ่มที่สร้าง กล่าวคือ ถ้า

- key = "uniform" จำนวนสุ่มที่สร้างขึ้นมาจะมีลักษณะการแจกแจงเอกรูปมีค่าอยู่ระหว่าง 0 ถึง 1 (เป็นค่าโดยปริยาย)
- key = "normal" จำนวนสุ่มที่สร้างขึ้นมาจะมีลักษณะการแจกแจงปรกติ (หรือแบบเกาส์เซียน) ที่มีค่าเฉลี่ยเท่ากับค่า 0 และมีค่าความแปรปรวน[22] (varaince) เท่ากับค่า 1

ตัวอย่างการใช้งานของคำสั่งนี้ เช่น

```
-->X = rand(2, 4, 'uniform')                    //สร้างเมทริกซ์สุ่มขนาด 2×4
 X  =
    0.3095371    0.9706916    0.0204748    0.3490364
    0.6762972    0.5441797    0.8941365    0.1105365

-->rand('normal')    //กำหนดให้ลักษณะการแจกแจงของจำนวนสุ่มเป็นแบบปรกติ

-->rand('info')      //ตรวจสอบดูว่า ณ ตอนนี้ คำสั่ง rand ใช้ลักษณะการแจกแจงแบบไหน
 ans  =
 normal

-->Y = rand(X, 'normal')                        //สร้างเมทริกซ์สุ่มขนาดเท่าเมทริกซ์ X
 Y  =
  - 0.2870145    0.9632817  - 0.9390319  - 0.4596412
  - 0.3563124    0.3714145    1.7104345    0.5145099
```

โดยทั่วไปเมื่อเปิดใช้งานโปรแกรม SCILAB แต่ละครั้ง สถานะของตัวกำเนิดจำนวนสุ่ม (random generator) จะถูกกำหนดให้อยู่ในสถานะตั้งต้นใหม่ทุกครั้ง ดังนั้นการใช้คำสั่ง rand(X) เพื่อสร้างเมทริกซ์สุ่มในครั้งแรกที่เปิดโปรแกรม SCILAB จะได้ผลลัพธ์เป็นค่าเดิมทุกครั้ง ถ้าต้องการเปลี่ยนสถานะตั้งต้นใหม่ให้กับตัวกำเนิดจำนวนสุ่มก็สามารถทำได้โดยใช้คำสั่ง

```
rand('seed', n)
```

ซึ่งเป็นคำสั่งที่ใช้กำหนดสถานะตั้งต้นของตัวกำเนิดจำนวนสุ่มให้มีค่าเท่ากับค่า n (ค่าโดยปริยาย คือ n = 0 สำหรับการเรียกใช้คำสั่ง rand ครั้งแรก) หากต้องการทราบว่าสถานะตั้งต้นของตัวกำเนิดจำนวนสุ่ม ณ ขณะนั้นมีค่าเท่าใดก็สามารถตรวจสอบดูได้จากการใช้คำสั่ง rand('seed') เช่น

[22] ค่าความแปรปรวน มีค่าเท่ากับค่าเบี่ยงเบนมาตรฐานยกกำลังสอง

```
-->rand('seed')
 ans  =
    1.473D+09
```

เมทริกซ์สุ่มมีประโยชน์มากโดยเฉพาะอย่างยิ่งในทางวิศวกรรมไฟฟ้าสื่อสาร เนื่องจากสัญญาณในระบบสื่อสารดิจิทัลส่วนใหญ่จะเป็นสัญญาณสุ่มที่มีลักษณะการแจกแจงเป็นแบบเกาส์เซียน ดังนั้นสัญญาณที่ต้องการสามารถที่จะถูกสร้างขึ้นได้จากเมทริกซ์สุ่มนี้

นอกจากนี้ถ้าต้องการสร้างเมทริกซ์สุ่มที่มีลักษณะการแจกแจงแบบอื่นๆ เช่น การแจกแจงทวินาม (binomial distribution), การแจกแจงเรขาคณิต (geometric distribution), การแจกแจงเอกรูป (uniform distribution), การแจกแจงไคกำลังสอง (chi-square distribution), การแจกแจงแกมมา (gamma distribution), การแจกแจงปัวซง (Poisson distribution) และการแจกแจงมาร์คอฟ (Markov distribution) เป็นต้น ก็สามารถทำได้โดยใช้คำสั่ง `grand` ตัวอย่างเช่น (ศึกษารายละเอียดรูปแบบการเรียกใช้งานคำสั่ง `grand` ได้จากคำสั่ง `help`)

```
-->Y = grand(2, 8, 'uin', 1, 9)        //สร้างเมทริกซ์สุ่มขนาด 2X8 ของเลขจำนวนเต็มระหว่าง
 Y  =                                  //ค่า 1 ถึง 9 โดยมีลักษณะการแจกแจงเอกรูป
    5.    1.    9.    6.    3.    7.    1.    2.
    7.    8.    1.    7.    6.    8.    8.    9.

-->Y = grand(3, 5, 'unf', -0.5, 1.5) //สร้างเมทริกซ์สุ่มขนาด 3X5 ของเลขจำนวนจริงระหว่าง
 Y  =                                  //ค่า -0.5 ถึง 1.5 โดยมีลักษณะการแจกแจงเอกรูป
    0.4010832    1.325155     1.3266747    0.5063800    0.5766849
    0.9409869  - 0.0420461    0.6165375    1.1516339    0.5931839
  - 0.3323572    0.5109970  - 0.1952440    0.4249484    1.4922694
```

4.6.5 เมทริกซ์ทแยงมุม

เมทริกซ์ทแยงมุม (diagonal matrix) เป็นเมทริกซ์ที่มีสมาชิกที่มีค่าตามที่กำหนดเฉพาะในแนวเส้นทแยงมุมหลัก ส่วนสมาชิกที่ตำแหน่งอื่นๆ จะมีค่าเป็นค่า 0 ในโปรแกรม SCILAB สามารถสร้างเมทริกซ์ทแยงมุมได้โดยใช้คำสั่ง `diag` ซึ่งมีลักษณะการใช้งานหลายรูปแบบ ดังนี้

```
diag(vm, [k])
```

- ถ้าพารามิเตอร์ vm เป็นเวกเตอร์ของค่าที่จะแสดงในแนวเส้นทแยงมุม ส่วนค่าพารามิเตอร์ k จะเป็นตัวบอกว่าเป็นแนวเส้นทแยงมุมไหน นั่นคือถ้า
  - k < 0 หมายถึง เส้นทแยงมุมลำดับที่ |k| ที่อยู่ใต้เส้นทแยงมุมหลัก
  - k = 0 หมายถึง เส้นทแยงมุมหลัก (เป็นค่าโดยปริยาย)
  - k > 0 หมายถึง เส้นทแยงมุมลำดับที่ k ที่อยู่เหนือเส้นทแยงมุมหลัก

  ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->diag([1 2 3])                    //ค่า 1, 2, และ 3 อยู่ที่เส้นทแยงมุมหลัก
 ans  =
    1.    0.    0.
    0.    2.    0.
    0.    0.    3.

-->diag([1 2 3], 1)                 //ค่า 1, 2, และ 3 จะอยู่เหนือเส้นทแยงมุมหลักหนึ่งลำดับ
 ans  =
    0.    1.    0.    0.
    0.    0.    2.    0.
    0.    0.    0.    3.
    0.    0.    0.    0.

-->diag([1 2 3], -2)                //ค่า 1, 2, และ 3 จะอยู่ใต้เส้นทแยงมุมหลักสองลำดับ
 ans  =
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    1.    0.    0.    0.    0.
    0.    2.    0.    0.    0.
    0.    0.    3.    0.    0.
```

- ถ้าพารามิเตอร์ vm เป็นเมทริกซ์ ผลลัพธ์ที่ได้จะเป็นเวกเตอร์แนวตั้งที่มีสมาชิกเป็นค่าที่อยู่ในแนวเส้นทแยงมุมลำดับที่ |k| ของเมทริกซ์ vm ตัวอย่างการใช้งานคำสั่ง เช่น

```
-->A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
 A  =
    1.    2.     3.     4.
    5.    6.     7.     8.
    9.    10.    11.    12.
```

```
-->y = diag(A)
 y  =
    1.
    6.
    11.

-->z = diag(A, 2)
 z  =
    3.
    8.
```

4.6.6 เมทริกซ์สามเหลี่ยมด้านล่าง

เมทริกซ์สามเหลี่ยมด้านล่าง (lower triangular matrix) สามารถสร้างได้โดยใช้คำสั่ง `tril(X, k)` ซึ่งเป็นฟังก์ชันที่ใช้ในการสร้างเมทริกซ์ใหม่ที่มีขนาดเท่ากับเมทริกซ์ `X` โดยจะดึงเฉพาะสมาชิกในบริเวณสามเหลี่ยมด้านล่างของเมทริกซ์ `X` มาใส่ ส่วนสมาชิกในตำแหน่งอื่นๆ จะมีค่าเป็นค่า `0` สำหรับพารามิเตอร์ `k` จะเป็นตัวบอกว่าให้เริ่มดึงข้อมูลในแนวเส้นทแยงมุมไหน กล่าวคือมีลักษณะการใช้งานเหมือนพารามิเตอร์ `k` ที่ใช้ในเมทริกซ์ทแยงมุม นั่นคือ `k = 0` หมายถึงแนวเส้นทแยงมุมหลัก (เป็นค่าโดยปริยาย) ลองศึกษาการใช้งานคำสั่ง `tril` จากตัวอย่างต่อไปนี้จะได้เข้าใจลักษณะการใช้งานของคำสั่ง `tril` มากขึ้น

```
-->A = [1 2 3 4; 5 6 7 8; 9 10 11 12];

-->B = tril(A)                  //ดึงสมาชิกทั้งหมดที่อยู่ในแนวตั้งแต่เส้นทแยงมุมหลักลงมา
 B  =                           //ของเมทริกซ์ A มาสร้างเป็นเมทริกซ์ B
    1.    0.    0.    0.
    5.    6.    0.    0.
    9.    10.   11.   0.

-->C = tril(A, 2)               //ดึงสมาชิกทั้งหมดที่อยู่ในแนวตั้งแต่เส้นทแยงมุมที่อยู่เหนือ
 C  =                           //เส้นทแยงมุมหลักขึ้นไปสองลำดับลงมา
    1.    2.    3.    0.
    5.    6.    7.    8.
    9.    10.   11.   12.

-->D = tril(A, -1)              //ดึงสมาชิกทั้งหมดที่อยู่ในแนวตั้งแต่เส้นทแยงมุมที่อยู่ใต้
```

```
 D  =                                          //เส้นทแยงมุมหลักขึ้นไปหนึ่งลำดับลงมา
    0.    0.    0.    0.
    5.    0.    0.    0.
    9.    10.   0.    0.
```

4.6.7 เมทริกซ์สามเหลี่ยมด้านบน

เมทริกซ์สามเหลี่ยมด้านบน (upper triangular matrix) สามารถสร้างได้โดยใช้คำสั่ง `triu(X, k)` ซึ่งเป็นฟังก์ชันที่ใช้ในการสร้างเมทริกซ์ใหม่ที่มีขนาดเท่ากับเมทริกซ์ `X` โดยจะดึงเฉพาะสมาชิกในบริเวณสามเหลี่ยมด้านบนของเมทริกซ์ `X` มาใส่ ส่วนสมาชิกอื่นๆ จะมีค่าเป็นค่า `0` คำสั่งนี้มีรูปแบบการใช้งานเหมือนกับคำสั่ง `tril(X, k)` ทุกประการ เพียงแต่ผลลัพธ์ที่ได้จะเป็นเมทริกซ์สามเหลี่ยมด้านบน ตัวอย่างเช่น

```
-->A = [1 2 3 4; 5 6 7 8; 9 10 11 12];
-->triu(A)
 ans  =
    1.    2.    3.     4.
    0.    6.    7.     8.
    0.    0.    11.    12.
-->triu(A, 2)
 ans  =
    0.    0.    3.    4.
    0.    0.    0.    8.
    0.    0.    0.    0.
-->triu(A, -1)
 ans  =
    1.    2.     3.     4.
    5.    6.     7.     8.
    0.    10.    11.    12.
```

4.6.8 เมทริกซ์รูปแบบพิเศษ

เมทริกซ์รูปแบบพิเศษ ได้แก่ เมทริกซ์กล (magic matrix), อินเวอร์สของเมทริกซ์ฮิลเบิร์ต (Hilbert matrix), และเมทริกซ์ Franck โดยเมทริกซ์เหล่านี้สามารถสร้างได้จากการใช้คำสั่ง `testmatrix` ซึ่งมีรูปแบบการเรียกใช้งานดังนี้

```
testmatrix('matrix_name', n)
```

โดยที่พารามิเตอร์ matrix_name มีได้ 3 รูปแบบ คือ

1) magi สร้างเมทริกซ์จัตุรัสขนาด n×n ที่มีผลรวมของตัวเลขจำนวนเต็มในแนวนอน, แนวตั้ง, และแนวเส้นทแยงมุม เท่ากัน

2) frk สร้างเมทริกซ์ Franck

3) hilb สร้างอินเวอร์สของเมทริกซ์ฮิลเบิร์ตขนาด n×n โดยที่สมาชิกแถวที่ i และแนวตั้งที่ j ของเมทริกซ์ฮิลเบิร์ตมีค่าเท่ากับ H(i,j) = 1/(i+j-1)

ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->n = 3;

-->testmatrix('magi',n)              //สร้างเมทริกซ์กล
 ans  =
    8.    1.    6.                   //ผลรวมของตัวเลขจำนวนเต็มในแนวนอน, แนวตั้ง,
    3.    5.    7.                   //และแนวเส้นทแยงมุม มีค่าเท่ากับ 15
    4.    9.    2.

-->testmatrix('hilt', n)             //สร้างอินเวอร์ของเมทริกซ์ฮิลเบิร์ต
 ans  =
    9.   - 36.     30.
  - 36.    192.  - 180.
    30.  - 180.    180.

-->inv(testmatrix('hilt', n))        //สร้างเมทริกซ์ฮิลเบิร์ต
 ans  =
    1.           0.5          0.3333333
    0.5          0.3333333    0.25
    0.3333333    0.25         0.2
```

4.7 ตัวอย่างการคำนวณ

ในส่วนนี้จะยกตัวอย่างการแก้ไขปัญหาโจทย์ทางคณิตศาสตร์โดยใช้โปรแกรม SCILAB ดังต่อไปนี้

ตัวอย่างที่ 1 จงหาค่าของฟังก์ชัน $f(x) = \sin^2(\theta)\sec(\theta)$ เมื่อ $\theta = 0, \pi/8,$ และ $\pi/4$

วิธีทำ คำตอบของโจทย์ข้อนี้หาได้จากการใช้ชุดคำสั่งดังนี้

```
-->theta = [0, %pi/8, %pi/4];
-->fx = (sin(theta).^2) ./ cos(theta)
 fx  =
    0.     0.1585127     0.7071068
```

นั่นคือ f(x) มีค่าเท่ากับ 0, 0.1585127, และ 0.7071068 เมื่อ $\theta = 0, \pi/8,$ และ $\pi/4$ เรเดียน

ตัวอย่างที่ 2 จงหาค่าของ $f(x) = \sqrt{\tan(x)} + \log_{10}(\cosh(x))$ เมื่อ $x = 0, 0.5,$ และ 1

วิธีทำ ค่าของ x หาได้จาก

```
-->x = [0 0.5 1];
-->fx = sqrt(tan(x)) + log10(cosh(x))
 fx  =
    0.     0.7912879     1.43635
```

ดังนั้น f(x) มีค่าเท่ากับ 0, 0.7912879, และ 1.43635 เมื่อ x = 0, 0.5, และ 1

ตัวอย่างที่ 3 กำหนดให้ $\sin(x + 20^\circ) = 0.8$ จงหาค่าของ $\cot(70^\circ - x)$

วิธีทำ จากที่โจทย์กำหนดจะได้ว่า

$$\sin^{-1}(\sin(x + 20^\circ)) = \sin^{-1}(0.8)$$

$$x = \sin^{-1}(0.8) - \frac{(20^\circ)\pi}{180^\circ} \quad \text{เรเดียน}$$

ใช้โปรแกรม SCILAB หาค่า x ได้ดังนี้

```
-->x = asin(0.8) - 20*%pi/180
 x  =
    0.5782294                                    //หน่วยเป็นเรเดียน
```

จากนั้นแทนค่า x ลงใน $\cot\left(70^\circ - x\right) = \frac{1}{\tan\left(70^\circ - x\right)}$ จะได้ผลลัพธ์เท่ากับ

```
-->y = 1/tan((70*%pi/180) - x)              //แปลง 70° ให้มีหน่วยเป็นเรเดียน
 y  =
    1.3333333
```

นั่นคือถ้า $\sin\left(x + 20^\circ\right) = 0.8$ แล้วจะได้ว่า $\cot\left(70^\circ - x\right) = 1.3333333 = \frac{4}{3}$

ตัวอย่างที่ 4 จงหาค่า θ ที่ทำให้สมการ $\cos^3(\theta) - \cos^2(\theta) - 4\cos(\theta) + 4 = 0$

วิธีทำ กำหนดให้ $x = \cos(\theta)$ จะได้ว่า $x^3 - x^2 - 4x + 4 = 0$ ดังนั้นคำตอบของสมการนี้หาได้จาก

```
-->x = poly(0, 'x');
-->y = x^3 - x^2 - 4*x + 4;
-->x = roots(y)
 x  =
    1.
    2.
  - 2.
```

เนื่องจาก $x = \cos(\theta)$ ดังนั้นค่า θ สามารถหาได้จากคำสั่ง $\theta = \text{acos}(x)$ ดังนี้

```
-->theta = acos(x)
 theta  =
    0
    1.3169579i
    3.1415927 - 1.3169579i
```

```
-->y = clean(cos(theta ).^3 - cos(theta ).^2 - 4*cos(theta ) + 4)
 y  =
    0
    0
    0
```

คำตอบของสมการคือ θ = 0, 1.3169579i, และ 3.1415927-1.3169579i

ตัวอย่างที่ 5 กำหนดให้คะแนนสอบวิชาการสื่อสารดิจิทัลของนักศึกษาจำนวน 30 คนมีดังนี้ (จากคะแนนเต็ม 100 คะแนน)

68	45	93	76	80	85	78	85	88	90
50	55	65	68	70	90	75	82	67	45
88	90	64	78	75	59	60	73	78	65

จงหาค่าต่อไปนี้

1) ค่าคะแนนต่ำสุดของนักศึกษาในห้อง และมีนักศึกษากี่คนที่ได้คะแนนต่ำสุด
2) ค่าเฉลี่ยเลขคณิต, ค่าเฉลี่ยเรขาคณิต, และค่าเฉลี่ยฮาร์มอนิก ของคะแนนในห้อง
3) จงเรียงลำดับคะแนนของนักศึกษาในห้องจากค่ามากไปหาค่าน้อย

วิธีทำ การแก้ปัญหาโจทย์ข้อนี้ให้เริ่มต้นจากการนำข้อมูลคะแนนสอบของนักศึกษาจำนวน 30 คนมาจัดให้อยู่ในรูปของเวกเตอร์ดังนี้

```
-->x = [68 45 93 76 80 85 78 85 88 90 ...
-->     50 55 65 68 70 90 75 82 67 45 ...
-->     88 90 64 78 75 59 60 73 78 65];
-->
```

จากนั้นก็สามารถหาคำตอบที่ต้องการได้ดังต่อไปนี้

1) คะแนนต่ำสุดและจำนวนนักศึกษาที่ได้คะแนนต่ำสุด สามารถหาได้จาก

```
-->xmin = min(x)                    //หาคะแนนต่ำสุด
 xmin  =
    45.
```

```
-->index = find(x == xmin)      //หาตำแหน่งของข้อมูลที่มีค่าเท่ากับคะแนนต่ำสุด
 index  =
    2.    20.

-->Nmin = length(index)         //หาจำนวนนักศึกษาที่ได้คะแนนต่ำสุด
 Nmin  =
    2.
```

นั่นคือคะแนนต่ำสุดมีค่าเท่ากับ 45 และมีนักศึกษาจำนวน 2 คนที่ได้คะแนนต่ำสุด

2) ค่าเฉลี่ยเลขคณิต, ค่าเฉลี่ยเรขาคณิต, และค่าเฉลี่ยฮาร์มอนิก สามารถหาได้จาก

```
-->am = mean(x)
 am  =
    72.833333

-->gm = geomean(x)
 gm  =
    71.501303

-->hm = harmean(x)
 hm  =
    70.047351
```

ผลลัพธ์ที่ได้คือ ค่าเฉลี่ยเลขคณิตมีค่าเท่ากับ 72.833333, ค่าเฉลี่ยเรขาคณิตมีค่าเท่ากับ 71.501303, และค่าเฉลี่ยฮาร์มอนิกมีค่าเท่ากับ 70.047351

3) จงเรียงลำดับคะแนนของนักศึกษาในห้องจากค่ามากไปหาค่าน้อย สามารถหาได้จาก

```
-->MaxToMin = sort(x)
 MaxToMin  =
         column 1 to 9
    93.    90.    90.    90.    88.    88.    85.    85.    82.
         column 10 to 18
    80.    78.    78.    78.    76.    75.    75.    73.    70.
         column 19 to 27
    68.    68.    67.    65.    65.    64.    60.    59.    55.
         column 28 to 30
    50.    45.    45.
```

4.8 สรุป

ในบทนี้ได้อธิบายถึงฟังก์ชันพื้นฐานที่ใช้สำหรับการคำนวณทางคณิตศาสตร์ เช่น ฟังก์ชันพื้นฐานที่เกี่ยวกับตัวเลข, ฟังก์ชันตรีโกณมิติ, ฟังก์ชันไฮเพอร์โบลิก, และฟังก์ชันพื้นฐานทางสถิติ เป็นต้น ซึ่งเป็นประโยชน์มากในการแก้ไขปัญหาทางวิศวกรรมและวิทยาศาสตร์ โดยเมื่อเข้าใจถึงรูปแบบการใช้งานของฟังก์ชันต่างๆ แล้ว ก็จะทำให้สามารถพัฒนาโปรแกรมที่มีการคำนวณที่ซับซ้อนได้ นอกจากฟังก์ชันที่กล่าวมาในบทนี้ โปรแกรม SCILAB ยังได้เตรียมฟังก์ชันอื่นๆ อีกจำนวนมากที่ใช้ในการคำนวณเฉพาะทาง สำหรับผู้สนใจสามารถศึกษารายละเอียดเพิ่มเติมได้จากคำสั่ง `help`

4.9 แบบฝึกหัดท้ายบท

4.1 จงคำนวณหาค่าของฟังก์ชัน f(x) ต่อไปนี้ เมื่อ x = –2, 0, 2, –2i, และ 2i

4.1.1) $f(x) = \sqrt{x^3 - 2x} + \log_{10}\left(\left|3x - 5\right|\right)$

4.1.2) $f(x) = 3e^{(x+3)} + \log_e\left(\frac{e^x + e^{-x}}{2}\right)$

4.1.3) $f(x) = 4\cosh^3(x) - 3\cosh(x)$

4.2 กำหนดให้ตัวแปร x เป็นเลขจำนวนจริง จงพิสูจน์ว่า

4.2.1) $\log_a(x) = \log(x)/\log(a)$

4.2.2) $\log_a(xy) = \log_a(x) + \log_a(y)$

4.2.3) $\log_a(x)^y = y\log_a(x)$

4.3 กำหนดให้ตัวแปร x เป็นเลขจำนวนจริง จงพิสูจน์ว่า

4.3.1) $\sin(2x) = 2\sin(x)\cos(x)$

4.3.2) $\cos(2x) = \cos^2(x) - \sin^2(x) = 1 - 2\sin^2(x) = 2\cos^2(x) - 1$

4.3.3) $\tan(2x) = \frac{2\tan(x)}{1 - \tan^2(x)}$

4.3.4) $\sin(3x) = 3\sin(x) - 4\sin^3(x)$

4.3.5) $\cos(3x) = 4\cos^3(x) - 3\cos(x)$

4.3.6) $\tan(3x) = \dfrac{3\tan(x) - \tan^3(x)}{1 - 3\tan^2(x)}$

4.3.7) $\sinh(3x) = 3\sinh(x) + 4\sinh^3(x)$

4.3.8) $\cosh(3x) = 4\cosh^3(x) - 3\cosh(x)$

4.3.9) $\tanh(3x) = \dfrac{3\tanh(x) + \tanh^3(x)}{1 + 3\tanh^2(x)}$

4.3.10) $r^z(\cos(x) + i\sin(x))^z = r^z\{\cos(zx) + i\sin(zx)\}$ โดยที่ r และ z เป็นเลขจำนวนจริง และ $i = \sqrt{-1}$

4.4 จงหารากหรือคำตอบของสมการต่อไปนี้

4.4.1) $x^2 - 2x + 5 = 0$

4.4.2) $x^3 - 6x^2 + 11x - 6 = 0$

4.4.3) $x^3 - 4x^2 + 6x - 4 = 0$

4.4.4) $x^7 - 14x^5 + 49x^3 - 36x = 0$

4.4.5) $x^7 + 2x^6 + 3x^5 - 26x^4 - 2x^3 + 72x^2 - 104x = 0$

4.5 กำหนดให้ $\{x_1, x_2, \ldots, x_n\}$ เป็นเลขจำนวนจริงหรือเลขจำนวนเชิงซ้อน จงพิสูจน์ว่า

4.5.1) $\overline{x_1 + x_2 + \ldots + x_n} = \overline{x}_1 + \overline{x}_2 + \ldots + \overline{x}_n$ เมื่อ $\overline{x}$ คือ ค่าสังยุคของ x

4.5.2) $\overline{x_1 \cdot x_2 \cdots x_n} = \overline{x}_1 \cdot \overline{x}_2 \cdots \overline{x}_n$

4.5.3) $\overline{x^n} = (\overline{x})^n$

4.5.4) $|x_1 + x_2|^2 \le |x_1|^2 + 2|x_1| \cdot |x_2| + |x_2|^2$

4.5.5) $|x_1 + x_2| \le |x_1| + |x_2|$

4.6 กำหนดให้คะแนนสอบวิชาการเขียนโปรแกรมภาษา SCILAB ของนักเรียนในห้องจำนวน 45 คนมีดังนี้ (จากคะแนนเต็ม 10 คะแนน)

8.3	7.2	8.9	9.2	9.6	5.1	7.4	7.5	8.1	5.8	6.9	8.4	7.8	9.3	9.5
5.3	5.8	7.3	6.9	8.4	8.1	7.9	7.5	8.5	6.5	5.5	7.8	7.7	9.2	8.5
7.1	9.7	7.8	8.2	7.2	9.1	8.7	8.2	7.4	8.3	7.6	8.9	8.7	8.4	7.6

จงหาค่าต่อไปนี้

4.6.1) ค่าคะแนนต่ำสุดของนักเรียนในห้อง และมีนักเรียนกี่คนที่ได้คะแนนต่ำสุด

4.6.2) ค่าคะแนนสูงสุดของนักเรียนในห้อง และมีนักเรียนกี่คนที่ได้คะแนนสูงสุด

4.6.3) ค่าเฉลี่ยของคะแนนในห้องค่าเฉลี่ยเลขคณิตและค่าเฉลี่ยเรขาคณิตของคะแนนในห้อง

4.6.4) ค่ามัธยฐานของคะแนนในห้อง

4.6.5) ค่าเบี่ยงเบนมาตรฐานของคะแนนในห้อง

4.6.6) จงวาดรูปฮิสโตแกรมเพื่อดูการแจกแจงของคะแนนสำหรับนักเรียนในห้องนี้

4.6.7) จงเรียงลำดับคะแนนของนักเรียนในห้องจากค่ามากไปหาค่าน้อย

บทที่ 5

การเขียนโปรแกรมด้วย SCILAB

ในบทนี้จะอธิบายถึงคำสั่งต่างๆ ที่ใช้บ่อยในการพัฒนาฟังก์ชันขึ้นมาใช้งาน ได้แก่ คำสั่งวนซ้ำ และคำสั่งทดสอบเงื่อนไข เป็นต้น รวมถึงหลักการเขียนโปรแกรมด้วย SCILAB เพื่อเป็นแนวทางให้ผู้อ่านสามารถพัฒนาฟังก์ชันใหม่ๆ ขึ้นมาใช้งานร่วมกับโปรแกรม SCILAB ได้ด้วยตนเองอย่างรวดเร็วและมีประสิทธิผล

5.1 คำสั่งวนซ้ำ

บ่อยครั้งในการเขียนโปรแกรมมีความจำเป็นที่จะต้องคำนวณชุดคำสั่งบางอย่างซ้ำเป็นจำนวนหลายๆ รอบซึ่งในกรณีนี้การใช้คำสั่งวนซ้ำจึงมีความจำเป็นมาก โดยโปรแกรม SCILAB ได้เตรียมคำสั่งสำหรับการวนซ้ำไว้อยู่สองรูปแบบ คือ คำสั่ง `for` และคำสั่ง `while` ซึ่งมีหลักการใช้งานดังนี้

5.1.1 คำสั่ง `for`

คำสั่ง `for` เหมาะสำหรับการใช้งานที่ต้องการให้โปรแกรมทำซ้ำชุดคำสั่งเดิมที่อยู่ภายในลูป (loop) เป็นจำนวนรอบตามที่กำหนดไว้ในนิพจน์ (expression) คำสั่ง `for` มีรูปแบบการใช้งาน ดังนี้

```
for variable = expression
      instruction_1;
            ⋮
      instruction_n;
end
```

กล่าวคือโปรแกรมจะทำซ้ำคำสั่ง (instruction) ทั้งหมดภายในลูปเป็นจำนวนรอบตามที่กำหนดโดยตัวแปรที่เป็นไปตามเงื่อนไขของนิพจน์ ตัวอย่างการใช้งานคำสั่ง `for` ตัวอย่างเช่น

```
-->L = 5;
-->x = [];
-->for i = 1:L
-->    x(i) = i;
-->end
```

ชุดคำสั่งนี้หมายความว่าเมื่อเริ่มต้นใช้งาน ตัวแปร `L` จะมีค่าเท่ากับ `5` และ `x` เป็นเมทริกซ์ว่าง (empty matrix) จากนั้นก็ทำการวนซ้ำโดยใช้ตัวแปร `i` เป็นตัวนับจำนวนซ้ำ นั่นคือตัวแปร `i` จะเริ่มจากค่า `1` แล้วเพิ่มขึ้นทีละ `+1` จนไปถึงค่า `5` โดยที่ค่าของตัวแปร `i` แต่ละค่าจะถูกบรรจุไว้ในสมาชิกลำดับที่ `i` ของเวกเตอร์ `x` ผลลัพธ์ของการประมวลผลชุดคำสั่งนี้คือ

```
-->x'
 ans  =
    1.    2.    3.    4.    5.
```

ลองศึกษาตัวอย่างต่อไปนี้สำหรับการประยุกต์การใช้งานคำสั่ง `for`

ตัวอย่างที่ 1 จงหาผลรวมของเลขจำนวนเต็มคี่ (odd number) ตั้งแต่ค่า `1` ไปเรื่อยๆ เป็นจำนวน `N = 10` ครั้ง โดยใช้โปรแกรม SCILAB นั่นคือการหาค่า `y` จากความสัมพันธ์

$$y = \underbrace{1+3+5+\ldots+(2N-1)}_{N \quad \text{elements}}$$

วิธีทำ จากโจทย์ที่ให้มา สามารถเขียนโปรแกรม SCILAB เพื่อคำนวณหาค่า `y` ได้ดังนี้

```
-->N = 10;
-->y = 0;
-->for i = 1:N
-->    y = y + (2*i - 1);
-->end
```

เมื่อประมวลผลชุดคำสั่งนี้แล้วจะได้ว่า ตัวแปร y มีค่าเท่ากับ 100 นั่นคือ

```
-->y
 y  =
    100.
```

5.1.2 คำสั่ง while

คำสั่ง while มีลักษณะการทำงานคล้ายกับคำสั่ง for เพียงแต่คำสั่ง while จะมีการทดสอบเงื่อนไขที่ผู้เขียนโปรแกรมกำหนดไว้ในนิพจน์ทุกๆ รอบของการวนซ้ำ กล่าวคือถ้าผลการทดสอให้ค่าตรรกะเป็นค่า 1 (เป็นจริง) โปรแกรมก็จะทำซ้ำชุดคำสั่งภายในลูปนั้นต่ออีกหนึ่งรอบ แต่ถ้าผการทดสอบให้ค่าตรรกะเป็นค่า 0 (เป็นเท็จ) โปรแกรมก็จะยกเลิกการทำงานชุดคำสั่งภายในลูปนั้นทันที คำสั่ง while มีรูปแบบการใช้งาน ดังนี้

```
while  expression
        instruction_1;
            ⋮
        instruction_n;
end
```

ตัวอย่างการใช้งานคำสั่งการวนซ้ำ while มีดังนี้

```
-->L = 5;
-->x = [];
-->i = 1;
-->while i <= L
-->      x(i) = i;
-->      i = i + 1;
-->end
```

สังเกตจะพบว่าชุดคำสั่งนี้คล้ายกับตัวอย่างของการใช้คำสั่ง for แต่การใช้คำสั่ง while นั้นจะต้องกำหนดค่าเริ่มต้นของตัวแปร i ก่อน เพื่อที่จะได้นำค่า i ไปทำการเปรียบเทียบกับค่า L ตามเงื่อนไขที่กำหนด ผลลัพธ์ของการประมวลผลชุดคำสั่งนี้คือ

```
-->x'
 ans  =
    1.    2.    3.    4.    5.
```

ซึ่งมีค่าเท่ากันกับผลลัพธ์ที่ได้จากการใช้คำสั่ง for ดังนั้นจะเห็นได้ว่าคำสั่ง while และคำสั่ง for มีความคล้ายกันมาก เพียงแต่คำสั่ง while จะมีการทดสอบเงื่อนไขก่อนที่จะตัดสินใจว่าจะสั่งให้โปรแกรมทำงานชุดคำสั่งภายในลูปหรือไม่

5.2 คำสั่งทดสอบเงื่อนไข

คำสั่งทดสอบเงื่อนไขมีความจำเป็นมากสำหรับการเขียนโปรแกรมคอมพิวเตอร์ที่ซับซ้อน โปรแกรม SCILAB ได้เตรียมคำสั่งทดสอบเงื่อนไขไว้อยู่ 2 รูปแบบ คือ คำสั่ง if และคำสั่ง select-case ซึ่งมีหลักการใช้งาน ดังนี้

5.2.1 คำสั่ง if

คำสั่ง if จะทำการทดสอบเงื่อนไขความสัมพันธ์ว่าเป็นจริงหรือเป็นเท็จ โดยมีรูปแบบการใช้งานคือ

```
if expression then
    instruction_1;
        ⋮
    instruction_n;
end
```

กล่าวคือถ้าผลการทดสอบเงื่อนไขในนิพจน์เป็นจริง โปรแกรม SCILAB ก็จะทำคำสั่งทั้งหมดที่อยู่ระหว่างคำว่า `then` และ `end` แต่ถ้าผลการทดสอบเป็นเท็จ โปรแกรม SCILAB จะไม่ทำคำสั่งทั้งหมดที่อยู่ระหว่างคำว่า `then` และ `end`

นอกจากนี้คำสั่ง `if` ยังสามารถนำไปใช้งานกับการตัดสินใจที่ซับซ้อนมากขึ้นได้โดยการใช้งานร่วมกับ `else` ซึ่งมีรูปแบบการใช้งาน ดังนี้

```
if  expression  then
    instructions_set1;
else
    instructions_set2;
end
```

นั่นคือถ้าผลการทดสอบเงื่อนไขในนิพจน์เป็นจริง โปรแกรม SCILAB จะทำคำสั่งทั้งหมดที่อยู่ระหว่างคำว่า `then` และ `else` แต่ถ้าผลการทดสอบเป็นเท็จ โปรแกรม SCILAB จะทำคำสั่งทั้งหมดที่อยู่ระหว่างคำว่า `else` และ `end` แทน

ในการใช้งานที่มีการตัดสินใจที่ซับซ้อนมากยิ่งขึ้น ผู้ใช้ก็สามารถใช้งานคำสั่ง `if` ร่วมกับ `elseif` ได้ โดยมีรูปแบบการใช้งานดังนี้

```
if  expression_1  then
    instructions_set1;
elseif  expression_2  then
    instructions_set2;
else
    instructions_set3;
end
```

นั่นคือถ้าผลการทดสอบเงื่อนไขในนิพจน์ expression_1 เป็นจริง โปรแกรม SCILAB จะทำชุดคำสั่ง instructions_set1 แต่ถ้าผลการทดสอบเป็นเท็จ โปรแกรม SCILAB ก็จะทำการทดสอบเงื่อนไขในนิพจน์ expression_2 ต่อไปทันที โดยที่ถ้าผลการทดสอบในนิพจน์ expression_2 เป็นจริง โปรแกรม SCILAB ก็จะทำชุดคำสั่ง instructions_set2 แต่ถ้าผลการทดสอบเป็นเท็จ ก็จะทำชุดคำสั่ง instructions_set3

พิจารณาตัวอย่างโปรแกรมการสร้างเมทริกซ์จัตุรัสขนาด NxN (ถ้ากำหนดให้ N = 3) โดยสมาชิกทุกตัวที่อยู่ใต้เส้นทแยงมุมหลักมีค่าเท่ากับ 1 ส่วนสมาชิกทุกตัวที่อยู่ในแนวเส้นทแยงมุมหลักมีค่าเท่ากับ 2 และสมาชิกทุกตัวที่อยู่เหนือเส้นทแยงมุมหลักมีค่าเท่ากับ 3 ดังนั้นเมทริกซ์นี้สามารถเขียนเป็นชุดคำสั่งในโปรแกรม SCILAB ได้ คือ

```
-->N = 3;
-->A = [];
-->for i = 1:N
-->    for j = 1:N
-->        if i < j then
-->           A(i, j) = 3;
-->        elseif i == j then
-->           A(i, j) = 2;
-->        else
-->           A(i, j) = 1;
-->        end
-->    end
-->end
```

เมื่อทำการประมวลผลชุดคำสั่งนี้แล้วจะพบว่าผลลัพธ์ที่ได้คือ เมทริกซ์ตามที่กำหนด นั่นคือ

```
-->A
 A  =
    2.    3.    3.
    1.    2.    3.
    1.    1.    2.
```

5.2.2 คำสั่ง select-case

การทดสอบเงื่อนไขความสัมพันธ์สามารถทำได้อีกรูปแบบหนึ่งคือ การใช้คำสั่ง select ร่วมกับ case โดยมีรูปแบบการใช้งาน ดังนี้

```
select expression
        case  result_1 then instructions_1;
                    ⋮
        case  result_n then instructions_n;
        else  instruction_m;
end
```

กล่าวคือโปรแกรม SCILAB จะทำการทดสอบเงื่อนไขในนิพจน์ก่อน ถ้าผลการทดสอบตรงกับข้อกำหนดของ case ใด โปรแกรม SCILAB ก็จะทำชุดคำสั่งทั้งหมดที่อยู่ภายใน case นั้น[23] แต่ถ้าผลการทดสอบไม่ตรงกับข้อกำหนดของ case ใดๆ โปรแกรม SCILAB ก็จะทำชุดคำสั่งทั้งหมดที่อยู่ระหว่าง else และ end (ในที่นี้คือชุดคำสั่ง instruction_m) ตัวอย่างเช่น

```
-->N = 7;
-->x = [];
-->for i = 1:N
-->    select i
-->        case 1 then x(1) = 2;
-->        case 2 then x(2) = 3;
-->        case 3 then x(3) = 4;
-->        else x(i) = 5;
-->    end
-->end
```

[23] ในการใช้คำสั่ง select คำว่า “then” จะ**ต้อง**อยู่ในบรรทัดเดียวกันกับคำว่า “case” เสมอ

โดยผลลัพธ์ที่ได้จากการประมวลผลชุดคำสั่งนี้ คือ

```
-->x'
 ans  =
    2.    3.    4.    5.    5.    5.    5.
```

5.3 การพัฒนาฟังก์ชัน

ในส่วนนี้จะอธิบายถึงหลักการพัฒนาฟังก์ชัน (function) ขึ้นมาใช้งานร่วมกับโปรแกรม SCILAB โดยมีรายละเอียดดังนี้

5.3.1 ไฟล์สคริปต์และไฟล์ฟังก์ชัน

ไฟล์สคริปต์ (file script) และไฟล์ฟังก์ชัน (file function) เป็นสิ่งที่มีประโยชน์มากในการพัฒนาโปรแกรม โดยทั่วไปไฟล์สคริปต์และไฟล์ฟังก์ชันมีข้อแตกต่างกันดังต่อไปนี้

5.3.1.1 ไฟล์สคริปต์

ไฟล์สคริปต์เป็นไฟล์ข้อมูล (text file) ที่รวบรวมคำสั่งต่างๆ ที่ต้องการให้โปรแกรม SCILAB ทำงานบรรจุไว้ในไฟล์เดียว เพื่อที่จะได้สะดวกต่อการทำงานที่ต้องการแบ่งแยกส่วนของขั้นตอนการทำงานแต่ละขั้นตอนให้ชัดเจน จะได้ง่ายต่อการตรวจสอบ ปรับปรุง และแก้ไขขั้นตอนการทำงานนั้นๆ ดังนั้นไฟล์สคริปต์จึงช่วยหลีกเลี่ยงปัญหาความผิดพลาดที่อาจจะเกิดขึ้นได้ ถ้านำเอาชุดคำสั่งทั้งหมดมารวมไว้ในโปรแกรมหลักที่เดียว นอกจากนี้การตรวจสอบและการแก้ไขข้อมูลต่างๆ ในไฟล์สคริปต์ก็สามารถทำได้ง่าย โดยการใช้โปรแกรมเอดิเตอร์ทั่วไป เช่น โปรแกรม Notepad, WordPad, หรืออาจจะใช้โปรแกรมเอดิเตอร์ของ SCILAB เองที่เรียกว่า “Scipad” ก็ได้

ในทางปฏิบัติไฟล์สคริปต์จะเป็นไฟล์ที่ปิดท้ายด้วย `.sce` และสามารถใช้คำสั่ง `exec` ในการประมวลผลไฟล์สคริปต์ การทำงานของไฟล์สคริปต์จะเทียบเท่ากับการป้อนคำสั่งผ่านโปรแกรม SCILAB โดยตรงตามปรกติทีละคำสั่งตามลำดับที่ปรากฏอยู่ในไฟล์สคริปต์ ดังนั้นตัวแปรทั้งหมดที่ใช้ในไฟล์สคริปต์จึงถือว่าเป็น ตัวแปรโกลบอล (global variable) ซึ่งหมายความว่า ถ้าตัวแปรต่างๆ ที่ใช้ในหน้าต่างคำสั่งก่อนการเรียกใช้งานไฟล์สคริปต์มีชื่อซ้ำกับตัวแปรที่อยู่ในไฟล์สคริปต แล้ว ค่าของตัวแปรเหล่านั้นก็สามารถที่จะถูกเปลี่ยนไปได้ตามคำสั่งที่อยู่ในไฟล์สคริปต์

5.3.1.2 ไฟล์ฟังก์ชัน

ถ้าต้องการสร้างไฟล์ที่ทำหน้าที่เสมือนเป็นฟังก์ชันหนึ่งในตัวโปรแกรม SCILAB ขึ้นมาใช้งานเอง โดยมีลักษณะการทำงานเหมือนกับฟังก์ชันในตัว (built-in function) ของโปรแกรม SCILAB กล่าวคือมีการส่งผ่านค่าของตัวแปรต่างๆ ไปให้กับฟังก์ชัน จากนั้นฟังก์ชันก็จะนำค่าของตัวแปรเหล่านี้ไปประมวลผลตามคำสั่งที่มีอยู่ภายในตัวฟังก์ชัน แล้วก็ส่งผลลัพธ์ที่ได้ออกมาจากตัวฟังก์ชัน ไฟล์ลักษณะนี้ในโปรแกรม SCILAB จะเรียกว่า ไฟล์ฟังก์ชันหรือ “Sci-file” ข้อดีของไฟล์ฟังก์ชันคือการช่วยทำให้ผู้ใช้งานโปรแกรม SCILAB สามารถพัฒนาฟังก์ชันรูปแบบใหม่ๆ ที่ไม่มีอยู่ในโปรแกรม SCILAB ขึ้นมาใช้งานเฉพาะด้านได้ตามความต้องการ ดังนั้นไฟล์ฟังก์ชันจึงมีประโยชน์มากในการเขียนโปรแกรมที่ซับซ้อน

ในทางปฏิบัติไฟล์ฟังก์ชันจะเป็นไฟล์ที่ปิดท้ายด้วย `.sci` และสามารถใช้คำสั่ง `exec` หรือ `getf` ในการประมวลผลไฟล์ฟังก์ชัน ข้อแตกต่างที่สำคัญระหว่างไฟล์สคริปต์และไฟล์ฟังก์ชันก็คือ ตัวแปรทั้งหมดที่ใช้ในไฟล์ฟังก์ชันจะถือว่าเป็น ตัวแปรโลคอล (local variable) ซึ่งหมายความว่า ถึงแม้ว่าตัวแปรต่างๆ ที่ใช้ในไฟล์ฟังก์ชันจะมีชื่อซ้ำกันกับตัวแปรที่ใช้อยู่ในหน้าต่างคำสั่ง (หรือนอกไฟล์ฟังก์ชัน) การเปลี่ยนแปลงค่าของตัวแปรที่ใช้อยู่ในหน้าต่างคำสั่งจะไม่มีผลกระทบต่อตัวแปรที่มีชื่อซ้ำกันที่ใช้อยู่ภายในไฟล์ฟังก์ชัน ดังนั้นผู้พัฒนาไฟล์ฟังก์ชันขึ้นมาใหม่จึงไม่ต้องกังวลกับการกำหนดชื่อตัวแปรที่ซ้ำกันกับตัวแปรที่ใช้อยู่ในหน้าต่างคำสั่ง หรือในไฟล์ฟังก์ชันอื่นๆ ซึ่งจะช่วยทำให้ผู้ใช้สามารถสร้างไฟล์ฟังก์ชันรูปแบบใหม่ๆ ขึ้นมาใช้งานได้อย่างเป็นอิสระจากผู้อื่น กล่าวคือผู้ใช้สามารถนำไฟล์ฟังก์ชันของผู้อื่นมาใช้กับงานของตนเองได้ เพียงแต่ป้อนค่าของตัวแปรต่างๆ ให้สอดคล้องกับรูปแบบและข้อกำหนดของไฟล์ฟังก์ชันนั้น

5.3.2 รูปแบบและข้อกำหนดในการเขียนไฟล์ฟังก์ชัน

โปรแกรม SCILAB มีฟังก์ชันสำเร็จรูปที่สามารถนำมาใช้งานมากมาย แต่ในกรณีที่ผู้ใช้ต้องการสร้างฟังก์ชันใหม่ๆ ขึ้นมาใช้งานเฉพาะทาง ก็สามารถทำได้เช่นกันโดยใช้โปรแกรมเอดิเตอร์ทั่วไปในการเขียนไฟล์ฟังก์ชัน แต่โครงสร้างของฟังก์ชันต้องสอดคล้องกับรูปแบบของโปรแกรม SCILAB ดังต่อไปนี้

```
function [y1, y2 , ..., yn] = function_name (x1, x2, ..., xm)
            instruction_1;
                 ⋮
            instruction_n;
endfunction
```

ข้อกำหนดในการเขียนไฟล์ฟังก์ชันมีดังนี้

- ไฟล์ฟังก์ชันที่สร้างขึ้นมาจะต้องเริ่มต้นด้วยคำว่า "function" เสมอ ส่วนคำปิดท้าย "endfunction" จะมีหรือไม่มีก็ได้
- ชื่อของไฟล์ฟังก์ชัน "function_name" จะต้องเหมือนกับชื่อของ SCI-file ทุกประการ (นั่นคือ ชื่อไฟล์จะต้องเป็น "function_name.sci" เท่านั้น)
- พารามิเตอร์ y1 ถึง yn จะเรียกว่า เอาต์พุตอาร์กิวเมนต์ (output argument) ซึ่งจะทำหน้าที่ในการส่งค่าผลลัพธ์ที่ได้จากการคำนวณภายในไฟล์ฟังก์ชันออกไปภายนอกไฟล์ฟังก์ชัน
- พารามิเตอร์ x1 ถึง xm จะเรียกว่า อินพุตอาร์กิวเมนต์ (input argument) ซึ่งจะทำหน้าที่ในการรับค่าตัวแปรจากภายนอกไฟล์ฟังก์ชันเข้ามาใช้ในการคำนวณภายในไฟล์ฟังก์ชัน
- จำนวนคำสั่ง (instruction) ที่ใช้ในไฟล์ฟังก์ชันจะมีจำนวนเท่าใดก็ได้

ตัวอย่างเช่น ถ้าต้องการสร้างฟังก์ชันเพื่อคำนวณหาผลรวมของเลขจำนวนคี่ตั้งแต่ค่า x1 ถึงค่า x2 ก็สามารถเขียนไฟล์ฟังก์ชันในโปรแกรม SCILAB เพื่อทำหน้าที่นี้ได้ดังนี้

```
function [OddSum] = MySum(x1, x2)
//[OddSum] = MySum(x1, x2) computes the summation of odd
//numbers between x1 and x2.

index = modulo(x1, x2);
if index == 1 then
   StartNumber = 1;
else
   StartNumber = x1 + 1;
end
```

```
OddSum = 0;
i = StartNumber;
while i <= x2
      OddSum = OddSum + i;
      i = i + 2;
end
endfunction
```

เมื่อเขียนไฟล์ฟังก์ชันเสร็จแล้วจะต้องบันทึกไฟล์นี้ในชื่อของ MySum.sci เท่านั้น ก่อนที่จะนำไฟล์ฟังก์ชันนี้มาใช้งานเสมือนหนึ่งว่าเป็นฟังก์ชันมาตรฐานของโปรแกรม SCILAB จะต้องทำการโหลด (load) ไฟล์ฟังก์ชันนี้ก่อน โดยใช้คำสั่ง exec หรือ getf ดังแสดงในตัวอย่างต่อไปนี้

```
-->y = MySum(1,10)
                !--error 4
undefined variable : MySum   //หมายความว่ายังไม่สามารถใช้งานฟังก์ชัน MySum ได้

-->exists('MySum')           //ตรวจสอบดูว่ามีตัวแปรชื่อ MySum ในพื้นที่ใช้งานหรือไม่
 ans  =
    0.                       //ค่า 0 หมายถึงไม่มีตัวแปรชื่อ MySum ในพื้นที่ใช้งาน

-->exec('MySum.sci');        //เทียบเท่ากับการใช้คำสั่ง getf('MySum.sci')

-->exists('MySum')
 ans  =
    1.                       //ค่า 1 หมายถึงมีตัวแปรชื่อ MySum ในพื้นที่ใช้งาน
```

คำสั่ง exists เป็นคำสั่งที่ใช้ในการยืนยันการมีอยู่จริง[24] (existence) ของชื่อตัวแปรนั้นๆ กล่าวคือ ถ้ามีชื่อตัวแปรนั้นอยู่ในพื้นที่ใช้งาน (workspace) จริง ก็จะให้ผลลัพธ์เป็นค่า 1 แต่ถ้าไม่มีชื่อตัวแปรนั้นอยู่ก็จะให้ผลลัพธ์เป็นค่า 0 เมื่อทำการโหลดไฟล์ฟังก์ชันโดยใช้คำสั่ง exec หรือ getf แล้ว โปรแกรม SCILAB ก็จะมองเห็นไฟล์ฟังก์ชันที่สร้างขึ้นมานี้เป็นเสมือนตัวแปรตัวหนึ่งที่สามารถเรียกมาใช้งานได้ทันที เช่น

[24] เมื่อทำการโหลดไฟล์ฟังก์ชันแล้ว โปรแกรม SCILAB จะถือว่าชื่อไฟล์ฟังก์ชันนั้นเป็นเสมือนตัวแปรตัวหนึ่งภายในโปรแกรม SCILAB

```
-->y = MySum(2, 10)                    //ในที่นี้จะหมายถึง x1 = 2 และ x2 = 10
 y  =
    24.                                //ผลลัพธ์คือ 3 + 5 + 7 + 9 = 24

-->y = MySum(1, 10)                    //ในที่นี้จะหมายถึง x1 = 1 และ x2 = 10
 y  =
    25.                                //ผลลัพธ์คือ 1 + 3 + 5 + 7 + 9 = 25
```

จะเห็นได้ว่าเมื่อป้อนอินพุตพารามิเตอร์ x1 และ x2 ให้กับไฟล์ฟังก์ชัน MySum แล้ว โปรแกรม SCILAB ก็จะทำการประมวลผลชุดคำสั่งภายในไฟล์ฟังก์ชัน MySum จากนั้นก็จะส่งผลลัพธ์ที่ได้ออกมาบรรจุไว้ในตัวแปร y ถ้าหากป้อนอินพุตพารามิเตอร์ไม่ครบตามที่กำหนดไว้ในไฟล์ฟังก์ชัน MySum โปรแกรม SCILAB ก็จะไม่สามารถประมวลผลไฟล์ฟังก์ชันนี้ได้

```
-->y = MySum(1)                        //ป้อนอินพุตพารามิเตอร์แค่ตัวเดียว คือ x1
 !--error 4
undefined variable : x2
at line       12 of function MySum called by :
y = MySum(1)
```

อย่างไรก็ตามถ้ามีการกำหนดค่าของตัวแปรที่มีชื่อซ้ำกันกับชื่อของอินพุตพารามิเตอร์ของไฟล์ฟังก์ชันก่อนที่จะมีการเรียกใช้งานไฟล์ฟังก์ชัน ไฟล์ฟังก์ชันนั้นก็จะนำค่าของตัวแปรที่มีชื่อซ้ำกันกับชื่อของอินพุตพารามิเตอร์มาใช้ในการประมวลผลภายในตัวฟังก์ชันอย่างอัตโนมัติโดยไม่จำเป็นต้องมีการป้อนอินพุตพารามิเตอร์นั้น ดังแสดงในตัวอย่างต่อไปนี้

```
-->x2 = 10
 x2  =
    10.
-->y = MySum(1)                        //กำหนดให้ x2 = 10 อัตโนมัติ
 y  =
    25.
```

ไฟล์ฟังก์ชันที่ถูกโหลดโดยใช้คำสั่ง exec หรือ getf จะทำหน้าที่เสมือนเป็นฟังก์ชันมาตรฐานของโปรแกรม SCILAB อย่างไรก็ตามถ้าป้อนคำสั่ง clear เข้าไปที่หน้าต่างคำสั่ง ตัวแปรและไฟล์ฟังก์ชันต่างๆ ก็จะถูกลบออกไปจากโปรแกรม SCILAB ดังแสดงในตัวอย่างต่อไปนี้

```
-->clear;

-->x2
  !--error 4
undefined variable : x2                    //ไม่มีชื่อตัวแปร x2 ในพื้นที่ใช้งาน

-->exists('MySum')
 ans  =
    0.                                     //ไม่มีชื่อไฟล์ฟังก์ชัน MySum ในพื้นที่ใช้งาน
```

5.3.3 การใช้งานคำสั่ง argn

จากตัวอย่างที่แสดงข้างต้น ในกรณีที่ผู้ใช้ลืมป้อนอินพุตพารามิเตอร์บางตัวให้กับไฟล์ฟังก์ชันที่สร้างขึ้นมา แต่ชื่อของอินพุตพารามิเตอร์เหล่านั้นไปซ้ำกับชื่อของตัวแปรที่เคยเรียกใช้อยู่ในพื้นที่ใช้งาน ไฟล์ฟังก์ชันก็จะนำเอาค่าของตัวแปรที่มีชื่อซ้ำกับชื่อของอินพุตพารามิเตอร์ไปใช้ในการประมวลผลภายในตัวไฟล์ฟังก์ชัน ซึ่งการทำงานในลักษณะนี้เป็นสิ่งที่ควรหลีกเลี่ยงเนื่องจากอาจจะเป็นการใช้ค่าของอินพุตพารามิเตอร์ที่ผิดจุดประสงค์ อันจะส่งผลทำให้โปรแกรมทั้งหมดเกิดความเสียหายได้ เพราะว่าใช้ค่าที่ไม่ถูกต้องในการคำนวณ

ดังนั้นโปรแกรม SCILAB จึงได้เตรียมคำสั่ง argn เพื่อให้ผู้พัฒนาไฟล์ฟังก์ชันสามารถตรวจสอบได้ว่า โปรแกรมภายนอกที่กำลังเรียกใช้ไฟล์ฟังก์ชันนี้ได้ป้อนอินพุตอาร์กิวเมนต์กี่ตัวและต้องการเอาต์พุตอาร์กิวเมนต์กี่ตัว เพื่อให้ผู้พัฒนาไฟล์ฟังก์ชันจะได้สามารถเขียนโปรแกรมให้ทำหน้าที่สอดคล้องกับจำนวนอาร์กิวเมนต์ที่รับเข้ามาและที่ส่งออกไป รูปแบบการใช้งานของคำสั่ง argn มีดังนี้

```
[lhs, rhs] = argn()
```

โดยที่พารามิเตอร์

- lhs เป็นเลขจำนวนเต็มบวกที่แสดงถึงจำนวนเอาต์พุตอาร์กิวเมนต์ของไฟล์ฟังก์ชัน
- rhs เป็นเลขจำนวนเต็มบวกที่แสดงถึงจำนวนอินพุตอาร์กิวเมนต์ของไฟล์ฟังก์ชัน

คำสั่ง argn นี้จะต้องถูกใช้ภายในตัวไฟล์ฟังก์ชันที่พัฒนาขึ้นมาเท่านั้น ให้พิจารณาตัวอย่างต่อไปนี้จะได้เข้าใจถึงการทำงานของคำสั่ง argn มากขึ้น

```
function [OddSum, EvenSum] = MySum2(x1, x2)
//[OddSum, EvenSum] = MySum2(x1, x2) finds the summation of
//odd numbers and even numbers between x1 and x2.
[lhs, rhs] = argn();     //คำนวณหาค่าจำนวนเอาต์พุตอาร์กิวเมนต์และอินพุตอาร์กิวเมนต์
if rhs < 2 then  x2 = 10;  end
if modulo(x1, 2) == 1 then
   StartOddNumber  = x1;
   StartEvenNumber = x1 + 1;
else
   StartOddNumber  = x1 + 1;
   StartEvenNumber = x1;
end
OddSum = 0;
i = StartOddNumber;
while i <= x2
      OddSum = OddSum + i;
      i = i + 2;
end
if lhs == 2 then
   EvenSum = 0;
   i = StartEvenNumber;
   while i <= x2
         EvenSum = EvenSum + i;
         i = i + 2;
   end
end
endfunction
```

ไฟล์ฟังก์ชันนี้จะหาผลบวกของเลขจำนวนคี่และจำนวนคู่ทั้งหมดที่อยู่ระหว่างค่า x1 และค่า x2 โดยผลบวกของเลขจำนวนคี่จะถูกบรรจุไว้ในตัวแปรชื่อ OddSum และผลบวกของเลขจำนวนคู่

จะถูกบรรจุไว้ในตัวแปรชื่อ EvenSum เมื่อพิจารณาดูข้อมูลภายในไฟล์ฟังก์ชันจะพบว่ามีการกำหนดจำนวนอินพุตอาร์กิวเมนต์และเอาต์พุตอาร์กิวเมนต์ โดยที่ถ้าจำนวนอินพุตอาร์กิวเมนต์น้อยกว่าสองตัว (นั่นคือไม่มีการป้อนค่า x2 ให้กับไฟล์ฟังก์ชัน MySum2) ไฟล์ฟังก์ชันนี้ก็จะกำหนดให้ x2 = 10 โดยอัตโนมัติ เช่นเดียวกันถ้าจำนวนเอาต์พุตอาร์กิวเมนต์น้อยกว่าสองตัว ไฟล์ฟังก์ชันนี้ก็จะไม่ทำการคำนวณหาค่าผลรวมของเลขจำนวนคู่ที่อยู่ระหว่างค่า x1 และค่า x2 ดังนั้นเมื่อเขียนไฟล์ฟังก์ชันนี้เสร็จแล้วก็ให้บันทึกไฟล์ฟังก์ชันนี้ในชื่อของ MySum2.sci และก่อนที่จะนำไฟล์ฟังก์ชันนี้มาใช้งานก็จะต้องทำการโหลดไฟล์ฟังก์ชันนี้ก่อน ดังแสดงในตัวอย่างต่อไปนี้

```
-->getf('MySum2.sci')
-->[y1, y2] = MySum2(1,10)
 y2  =
    30.
 y1  =
    25.

-->[y1, y2] = MySum2(1)      //ไฟล์ฟังก์ชัน MySum2 จะใช้ค่า x2 = 10 โดยอัตโนมัติ
 y2  =
    30.
 y1  =
    25.

-->y1 = MySum2(1,10)         //ไฟล์ฟังก์ชัน MySum2 จะไม่แสดงผลรวมของเลขจำนวนคู่
 y1  =
    25.

-->y1 = MySum2(1)            //ไฟล์ฟังก์ชัน MySum2 จะใช้ค่า x2 = 10 โดยอัตโนมัติ
 y1  =                       //แต่จะไม่แสดงผลรวมของเลขจำนวนคู่
    25.
```

5.3.4 การเขียนฟังก์ชันแบบอินไลน์

การพัฒนาฟังก์ชันรูปแบบใหม่ขึ้นมาใช้งานสามารถทำได้ตามที่อธิบายไว้ข้างต้น เช่น ถ้าต้องการสร้างฟังก์ชันเพื่อหาค่าสูงสุดของเลขสองจำนวน ก็สามารถเขียนฟังก์ชันในโปรแกรม SCILAB ได้ดังนี้

```
function [y] = MyMax(x1,x2)

if x1 >= x2  then
   y = x1
else
   y = x2;
end

endfunction
```

จากนั้นให้ทำการบันทึกฟังก์ชันนี้ลงในไฟล์ โดยที่ชื่อไฟล์จะต้องเป็นชื่อเดียวกันกับชื่อของฟังก์ชัน นั่นคือชื่อไฟล์ MyMax.sci นอกจากนี้ก่อนจะใช้งานฟังก์ชันนี้ได้ก็จะต้องทำการโหลดฟังก์ชันโดยใช้คำสั่ง exec หรือ getf ก่อน

สังเกตจะพบว่าวิธีการสร้างฟังก์ชันข้างต้นค่อนข้างจะเสียเวลาเนื่องจากมีการทำงานหลายขั้นตอน ในกรณีที่ฟังก์ชันที่พัฒนาขึ้นมามีขนาดเล็กและถูกใช้งานเฉพาะภายในโปรแกรมหลักเพียงโปรแกรมเดียว โปรแกรม SCILAB อนุญาตให้มีการกำหนดฟังก์ชันแบบอินไลน์ (in-line function) ที่หน้าต่างคำสั่งได้โดยตรง ดังแสดงในตัวอย่างต่อไปนี้ (ยังคงเริ่มต้นฟังก์ชันด้วยคำว่า function และจะ**ต้อง**ลงท้ายด้วยคำว่า endfunction เสมอ)

```
-->function [y] = MyMax(x1,x2), if x1>=x2 then y=x1, ...
-->else y=x2; end; endfunction;

-->y = MyMax(1,5)
 y  =
    5.
```

วิธีการนี้จะทำให้สามารถนำฟังก์ชันที่สร้างขึ้นมาไปใช้งานได้ทันที เนื่องจากโปรแกรม SCILAB จะทำการโหลดฟังก์ชันนี้โดยอัตโนมัติ (ไม่ต้องเรียกผ่านคำสั่ง exec หรือ getf) วิธีการสร้างฟังก์ชันแบบนี้จะสะดวกในกรณีที่จำนวนคำสั่งภายในฟังก์ชันมีไม่มาก นอกจากวิธีการนี้แล้ว ผู้ใช้ยังสามารถใช้คำสั่ง deff ในการสร้างฟังก์ชันแบบอินไลน์ได้เช่นกันซึ่งมีรูปแบบการใช้งาน คือ

```
deff('[เอาต์พุต]=function_name(อินพุต)', 'ชุดคำสั่ง')
```

ตัวอย่างการใช้งานคำสั่ง deff เช่น

```
-->deff('[y] = MyMax(x1,x2)', 'if x1>x2 then y = x1; ...
-->else y = x2; end');
-->y = MyMax(1,5)
 y  =
    5.
```

จะเห็นได้ว่าการสร้างฟังก์ชันใหม่ขึ้นมาใช้งานในโปรแกรม SCILAB สามารถทำได้หลายลักษณะ ทำให้สะดวกต่อการใช้งานตามที่ต้องการ

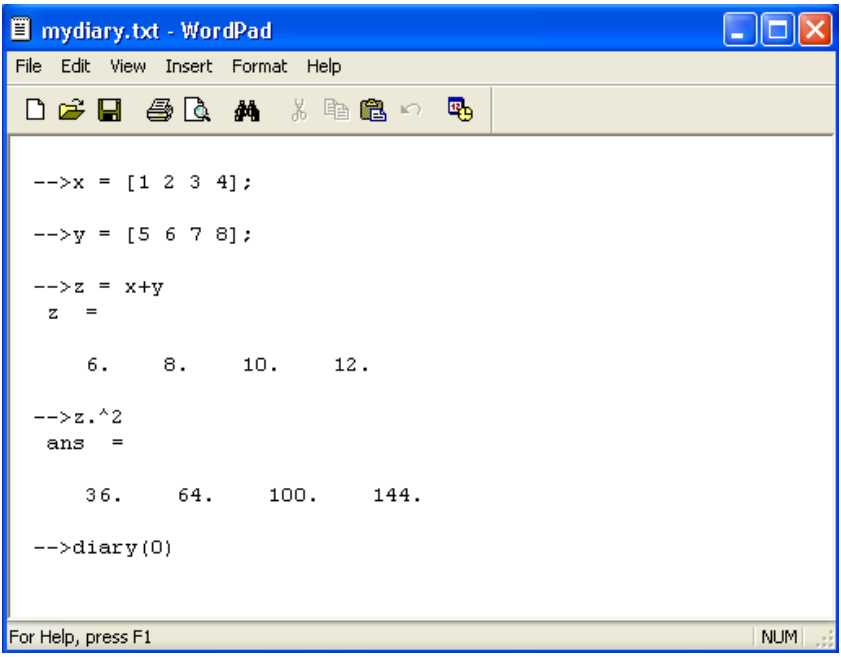
5.4 การเขียนไฟล์ไดอารี

ไฟล์ไดอารี (diary file) เป็นไฟล์ข้อมูลที่จะบันทึกข้อมูลทั้งหมดที่แสดงผลออกทางหน้าต่างคำสั่ง ตั้งแต่มีการเรียกใช้คำสั่ง diary ไปเรื่อยๆ จนถึงคำสั่ง diary(0) รูปแบบการเรียกใช้งานของคำสั่ง diary มีดังนี้

```
diary('output_filename')
```

โดยพารามิเตอร์ output_filename คือ ชื่อไฟล์ที่จะทำการบันทึกข้อมูลลงไป ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->diary('mydiary.txt')
-->x = [1 2 3 4];
-->y = [5 6 7 8];
-->z = x+y
 z  =
    6.    8.    10.    12.
-->z.^2
 ans  =
    36.    64.    100.    144.
-->diary(0)
```



รูปที่ 5.1 รายละเอียดของข้อมูลภายในไฟล์ `mydiary.txt` โดยใช้โปรแกรมเอดิเตอร์ WordPad

เมื่อใช้คำสั่ง `diary(0)` ก็จะถือว่าเป็นการสิ้นสุดการบันทึกข้อมูลและทำให้ได้ไฟล์เอาต์พุตออกมาที่ชื่อว่า "`mydiary.txt`" ในสารบบที่กำลังทำงานอยู่ ณ ขณะนั้น เมื่อลองใช้โปรแกรมเอดิเตอร์ที่ชื่อว่า "WordPad" เปิดอ่านไฟล์นี้ดูก็จะได้ตามที่แสดงในรูปที่ 5.1 ซึ่งจะพบว่าข้อมูลที่อยู่ในไฟล์นี้จะเหมือนกับข้อมูลที่ปรากฏอยู่ในหน้าต่างคำสั่ง ตั้งแต่มีการเรียกใช้คำสั่ง `diary` จนถึงคำสั่ง `diary(0)`

ประโยชน์ของไฟล์ไดอารีก็เพื่อทำหน้าที่เก็บบันทึกข้อมูลต่างๆ ที่แสดงผลออกทางหน้าต่างคำสั่ง เพื่อที่จะได้นำข้อมูลเหล่านี้ไปวิเคราะห์และประมวลผลในขั้นตอนต่อไป

5.5 ความรู้เพิ่มเติมในการเขียนไฟล์ฟังก์ชัน

ในส่วนนี้จะกล่าวถึงการโหลดไฟล์ฟังก์ชัน คำสั่งพื้นฐานที่ใช้บ่อยสำหรับการพัฒนาไฟล์ฟังก์ชัน พร้อมทั้งแนะนำเทคนิคการเขียนโปรแกรมให้มีประสิทธิภาพ

5.5.1 การโหลดไฟล์ฟังก์ชัน

ตอนนี้เป็นที่ทราบกันดีแล้วว่า ก่อนที่จะนำไฟล์ฟังก์ชันที่พัฒนาขึ้นมาใช้งานในโปรแกรม SCILAB ผู้ใช้จะต้องทำการโหลดไฟล์ฟังก์ชันนั้นก่อนโดยใช้คำสั่ง `exec` หรือ `getf` โดยไฟล์ฟังก์ชันที่ถูกโหลดเข้าไปในโปรแกรม SCILAB จะถือว่าเป็นตัวแปรตัวหนึ่งของโปรแกรม แต่ถ้ามีการใช้คำสั่ง `clear` ในหน้าต่างคำสั่ง ก็จะทำให้ไฟล์ฟังก์ชันนั้นถูกลบออกจากโปรแกรม SCILAB

โดยทั่วไปฟังก์ชันมาตรฐานต่างๆ ที่เกี่ยวข้องกันในโปรแกรม SCILAB จะถูกรวบรวมเก็บไว้ในไลบรารี (library) โดยไลบรารีที่โปรแกรม SCILAB เตรียมไว้ให้ ได้แก่ ไลบรารีสำหรับฟังก์ชันพื้นฐาน (`elemlib`), ไลบรารีทางด้านสถิติ (`statslib`), และไลบรารีสำหรับการประมวลผลสัญญาณ (`siglib`) เป็นต้น จะถูกเก็บไว้ในสารบบที่ชื่อว่า “SCIDIR/macros/” เมื่อ SCIDIR คือสารบบที่เก็บโปรแกรม SCILAB

ถ้าต้องการสร้างไลบรารีใหม่เพื่อเก็บไฟล์ฟังก์ชันต่างๆ ที่ผู้ใช้พัฒนาขึ้นมา ก็สามารถทำได้โดยใช้คำสั่ง `genlib` เพื่อที่ว่าไฟล์ฟังก์ชันเหล่านี้สามารถที่จะถูกเรียกใช้งานได้ตลอดเวลาเสมือนเป็นฟังก์ชันในตัว (built-in function) ของโปรแกรม SCILAB (ไม่ต้องมีการโหลดไฟล์ฟังก์ชันก่อนการเรียกใช้งาน) ทั้งนี้เป็นเพราะว่าโปรแกรม SCILAB จะทำการโหลดไลบรารีต่างๆ ทุกครั้งเมื่อเริ่มใช้งานโปรแกรม SCILAB สำหรับผู้สนใจสามารถศึกษารายละเอียดการใช้งานคำสั่ง `genlib` ได้จากคำสั่ง `help`

5.5.2 คำสั่งพื้นฐานสำหรับการพัฒนาฟังก์ชัน

ในส่วนนี้จะกล่าวถึงคำสั่งพื้นฐานที่ใช้บ่อยในการพัฒนาฟังก์ชัน ซึ่งมีดังต่อไปนี้

5.5.2.1 ฟังก์ชัน `disp`

คำสั่งนี้มีการเรียกใช้งานอยู่หลายรูปแบบ เช่น

- `disp(x)` เป็นคำสั่งที่ใช้ในการแสดงผลของตัวแปร `x` ตัวอย่างการใช้งาน เช่น

```
-->x = [2   4   6   8];
-->disp(x([1   3]))        //แสดงข้อมูลสมาชิกตัวที่หนึ่งและตัวที่สามของเวกเตอร์ x
    2.    6.
```

- `disp("string")` เป็นคำสั่งที่ใช้ในการแสดงข้อความในเครื่องหมาย `"..."` ออกมาที่หน้าต่างคำสั่ง

```
-->disp("Piya Kovintavewat")
   Piya Kovintavewat
```

นอกจากนี้คำสั่ง `disp` ยังมีการใช้งานในรูปแบบอื่นอีก เช่น

```
-->A = 3.5;  disp(A, "A = ")
 A =
    3.5
-->X = 3.5;  Y = 2;  disp(Y, "Y = ", X, "X = ")
 X =
    3.5
 Y =
    2.
-->Z = -5.2;  disp("Z = " + string(Z))
 Z = -5.2
```

5.5.2.2 ฟังก์ชัน `error`

รูปแบบการใช้งานของคำสั่งนี้คือ `error('string')` ซึ่งจะทำหน้าที่ในการแสดงข้อความที่อยู่ในเครื่องหมาย `'...'` ออกทางหน้าต่างคำสั่งและหยุดการทำงานของโปรแกรมนั้น

5.5.2.3 ฟังก์ชัน `warning`

รูปแบบการใช้งานของคำสั่งนี้คือ `warning('string')`ซึ่งจะทำหน้าที่ในการแสดงข้อความที่อยู่ในเครื่องหมาย `'...'` ออกทางหน้าต่างคำสั่งคล้ายกับคำสั่ง `error` แต่จะไม่มีการหยุดการทำงานของโปรแกรม

5.5.2.4 ฟังก์ชัน `pause`

เป็นคำสั่งที่ใช้ในการหยุดจังหวะการประมวลผลของโปรแกรม โดยโปรแกรม SCILAB จะแสดงเครื่องหมาย SCILAB prompt “`-->`” พร้อมทั้งแสดงระดับของการหยุดจังหวะ (level of the

pause) คำสั่งนี้มีประโยชน์มากสำหรับการตรวจสอบและแก้ไขโปรแกรม (debugging program) ตัวอย่างเช่น (ศึกษารายละเอียดเพิ่มเติมได้ในหัวข้อที่ 8.1.3)

```
-->pause                    //หยุดจังหวะการประมวลผลของโปรแกรม
-1->pause                   //ระดับที่หนึ่งของการหยุดจังหวะ
-2->pause                   //ระดับที่สองของการหยุดจังหวะ
-3->                        //ระดับที่สามของการหยุดจังหวะ
```

5.5.2.5 ฟังก์ชัน `abort`

เป็นคำสั่งที่ใช้ในการขัดจังหวะการประมวลผลของโปรแกรม โดยโปรแกรม SCILAB จะแสดงเครื่องหมาย SCILAB prompt "`-->`" และทำให้ระดับของการหยุดจังหวะลดลงจนเป็นศูนย์ ตัวอย่างเช่น (ศึกษารายละเอียดเพิ่มเติมได้ในหัวข้อที่ 8.1.4.1)

```
-->pause
-1->pause
-2->pause
-3->abort                   //ทำการขัดจังหวะการประมวลผลของโปรแกรม
-->                         //พร้อมทั้งทำให้ระดับของการหยุดจังหวะลดลงเป็นศูนย์
```

5.5.2.6 ฟังก์ชัน `break`

เป็นคำสั่งที่ใช้ในการหยุดการทำงานของลูป (loop) นั่นคือถ้าคำสั่งนี้อยู่ภายในชุดคำสั่ง `for` หรือ `while` จะส่งผลทำให้ลูปหยุดการทำงาน

5.5.2.7 ฟังก์ชัน `exit`

เป็นคำสั่งที่ใช้ในการหยุดการทำงานของโปรแกรม SCILAB นั่นคือหน้าต่างคำสั่งของโปรแกรม SCILAB ก็จะหายไปด้วย

5.5.2.8 ฟังก์ชัน quit

เป็นคำสั่งที่ใช้ในการหยุดการทำงานของโปรแกรม SCILAB หรือใช้การลดระดับของการหยุดจังหวะ (pause) ตัวอย่างเช่น

```
-->pause
-1->pause
-2->pause
-3->quit                    //ลดระดับของการหยุดจังหวะลงสองระดับ
-1->quit
-->
```

5.5.2.9 ฟังก์ชัน resume และ return

เป็นคำสั่งที่ทำให้โปรแกรมที่ถูกขัดจังหวะก่อนหน้านี้กลับมาทำงานต่ออีกครั้ง พร้อมทั้งสำเนา (copy) ตัวแปรโลคอลมาใช้ในพื้นที่ใช้งาน ณ ขณะนั้น (ศึกษารายละเอียดการใช้งานคำสั่ง resume และ return เพิ่มเติมได้ในหัวข้อที่ 8.1.2)

5.5.2.10 ฟังก์ชัน tic และ toc

ฟังก์ชัน tic เป็นคำสั่งที่ใช้ในการจับเวลาเริ่มต้นการทำงานของชุดคำสั่ง โดยทั่วไปจะใช้งานร่วมกับฟังก์ชัน toc ซึ่งจะทำหน้าที่ในการหยุดเวลาที่ตั้งไว้จากการใช้คำสั่ง tic พร้อมทั้งแสดงผลลัพธ์ของเวลาที่ใช้ไป (มีหน่วยเป็นวินาที) ออกทางหน้าต่างคำสั่ง ตัวอย่างเช่น

```
-->tic                      //เริ่มจับเวลา
-->deff('[y] = fsum(N)', 'y=0; for i=1:N, y=y+1; end');
-->y = fsum(2000000)
 y  =
    100.
-->toc                      //สิ้นสุดการจับเวลา ในที่นี้ได้ใช้เวลาไปประมาณ 5.719 วินาที
 ans  =
    5.719
```

5.5.3 ข้อแนะนำในการพัฒนาโปรแกรมให้มีประสิทธิภาพ

การพัฒนาโปรแกรมเพื่อให้สอดคล้องกับข้อกำหนดที่ต้องการสามารถที่จะทำได้หลายวิธี ทั้งนี้จะขึ้นอยู่กับทักษะและความถนัดของผู้พัฒนาโปรแกรมแต่ละคน โดยทั่วไปผู้พัฒนาโปรแกรมจะพยายามที่จะทำให้ได้โปรแกรมที่สามารถทำงานได้อย่างถูกต้องและรวดเร็ว

เนื่องจากโปรแกรม SCILAB เป็นโปรแกรมที่ทำงานบนพื้นฐานของเมทริกซ์ ดังนั้นหลักการเขียนโปรแกรมที่ดีคือ ควรจะพยายามใช้ประโยชน์จากโครงสร้างของเมทริกซ์และพยายามหลีกเลี่ยงการใช้คำสั่งวนซ้ำต่างๆ ให้มากที่สุด ลองพิจารณตัวอย่างการคำนวณหาค่าผลรวมของฟังก์ชัน $f(i) = \frac{3i^2+2i-1}{5i^2}$ สำหรับค่าตัวแปร i ตั้งแต่ค่า 1 ถึงค่า 200000 นั่นคือการหาค่าของตัวแปร y จาก

$$y = \sum_{i=1}^{200000} \left(\frac{3i^2+2i-1}{5i^2} \right)$$

ซึ่งสามารถเขียนเป็นชุดคำสั่งเพื่อหาค่า y โดยใช้คำสั่งวนซ้ำได้ดังนี้

```
-->y = 0;
-->tic                                  //เริ่มจับเวลา
-->for i=1:200000
-->   y = y + (3*i^2 + 2*i - 1)/(5*i^2);
-->end
-->toc                                  //หยุดจับเวลา และแสดงเวลาทั้งหมดที่ใช้
 ans  =
    2.937
-->y
 y  =
    120004.78
```

ชุดคำสั่งนี้มีการใช้คำสั่งวนซ้ำทำให้ใช้เวลาในการประมวลผลรวมทั้งสิ้น 2.937 วินาที อย่างไรก็ตามผู้ใช้สามารถที่จะพัฒนาชุดคำสั่งใหม่ให้มีผลลัพธ์เท่าเดิม แต่ใช้ความเร็วในการประมวลผลน้อยลงได้โดยใช้ประโยชน์จากโครงสร้างของเมทริกซ์ ดังต่อไปนี้

```
-->tic
-->i = 1:200000;
-->y = sum((3*i.^2 + 2*i - 1)./(5*i.^2));
-->toc
 ans  =
    0.234
-->y
 y  =
    120004.78
```

จะเห็นได้ว่าชุดคำสั่งนี้จะใช้เวลาในการประมวลผลเพียง 0.234 วินาที ซึ่งเร็วกว่าชุดคำสั่งเดิมมาก ดังนั้นผู้พัฒนาโปรแกรมควรที่จะใช้ประโยชน์จากโครงสร้างของเมทริกซ์ในการสร้างโปรแกรมให้มากที่สุด เพื่อที่จะทำให้ได้โปรแกรมที่สามารถทำงานได้อย่างรวดเร็วและมีประสิทธิผล

5.6 ตัวอย่างการคำนวณ

ในส่วนนี้จะยกตัวอย่างการแก้ไขปัญหาโจทย์ทางคณิตศาสตร์โดยใช้โปรแกรม SCILAB ดังต่อไปนี้

ตัวอย่างที่ 2 จงเขียนโปรแกรมเพื่อเรียงลำดับเลขจำนวนเต็มใดๆ 3 จำนวน เพื่อให้แสดงผลลัพธ์จากค่าน้อยไปหาค่ามาก โดยกำหนดให้มีรูปแบบการเรียกใช้งานดังนี้

```
y = MinToMax(a, b, c)
```

เมื่อ a, b, และ c คือเลขจำนวนเต็ม และ y คือเวกเตอร์ที่มีสมาชิกเรียงลำดับจากค่าน้อยไปหาค่ามาก

วิธีทำ โปรแกรมนี้สามารถสร้างได้ดังนี้

```
function y = MinToMax(a, b, c)
x = [a, b, c];
m1 = min(x);
x(find(x == m1)) = [];
m2 = min(x);
```

```
x(find(x == m2)) = [];
m3 = x;
y = [m1, m2, m3];
endfunction
```

เมื่อเขียนไฟล์ฟังก์ชันเสร็จก็ทำการบันทึกไฟล์นี้ในชื่อ MinToMax.sci ตัวอย่างการใช้งานเช่น

```
-->getf('MinToMax.sci')              //ทำการโหลดไฟล์ฟังก์ชันก่อนเรียกใช้งาน
-->y = MinToMax(3, -3, 1)
 y  =
  - 3.    1.    3.
-->y = MinToMax(-0.2, 99, 0.45)
 y  =
  - 0.2    0.45    99.
```

ตัวอย่างที่ 3 จงเขียนโปรแกรมเพื่อคำนวณหาระยะทางที่สั้นที่สุด d ระหว่างจุดสองจุดในระบบพิกัดคาร์ทีเซียน นั่นคือ P1(x1, y1) และ P2(x2, y2) โดยมีรูปแบบการเรียกใช้งาน คือ

```
d = FindDistance(x1, y1, x2, y2)
```

วิธีทำ เนื่องจากระยะทางระหว่างจุดสองจุดหาได้จาก $d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$ ดังนั้นโปรแกรมนี้สามารถสร้างได้ดังนี้

```
function [d] = FindDistance(x1, y1, x2, y2)
d = sqrt((x1-x2)^2 + (y1-y2)^2)
endfunction
```

เมื่อเขียนไฟล์ฟังก์ชันเสร็จก็ทำการบันทึกไฟล์นี้ในชื่อ FindDistance.sci ตัวอย่างเช่น

```
-->getf('FindDistance.sci')                    //ทำการโหลดไฟล์ฟังก์ชันก่อนเรียกใช้งาน

-->d = FindDistance(2, 3, 5, 7)
 d  =
    5.

-->d = FindDistance(0, 0, 5, 0)
 d  =
    5.

-->d = FindDistance(0, 0, 1, 1)
 d  =
    1.4142136
```

ตัวอย่างที่ 4 จงเขียนโปรแกรมเพื่อหาผลรวมของเลขจำนวนเต็มบวกตั้งแต่ n1 ถึง n2 ที่หารด้วยเลขจำนวนเต็มบวก m ลงตัว โดยกำหนดให้มีรูปแบบการเรียกใช้งานดังนี้

```
y = MySumMod(n1, n2, m)
```

วิธีทำ โปรแกรมนี้สามารถสร้างได้ดังนี้

```
function y = MySumMod(n1, n2, m)

x = n1:n2;
temp = modulo(x, m);
index = find(temp == 0);
y = sum(x(index))

endfunction
```

เมื่อเขียนไฟล์ฟังก์ชันเสร็จก็ทำการบันทึกไฟล์นี้ในชื่อ MySumMod.sci ตัวอย่างการใช้งานเช่น

```
-->getf('MySumMod.sci')                        //ทำการโหลดไฟล์ฟังก์ชันก่อนเรียกใช้งาน

-->y = MySumMod(1, 9, 2)
 y  =
    20.
```

```
-->y = MySumMod(1, 13, 3)
 y  =
    30.
-->y = MySumMod(3, 50, 7)
 y  =
    196.
```

ตัวอย่างที่ 5 จงเขียนโปรแกรมเพื่อหาค่าของฟังก์ชัน

$$f(x) = \begin{cases} 2x^2 - 3\cos(x) & ,x \geq 2 \\ \sqrt{\left(\cosh^2(x) + 1\right)} & ,-2 < x < 2 \\ \log_{10}(x) + e^x & ,x \leq -2 \end{cases}$$

เมื่อ x เป็นเลขจำนวนจริง โดยกำหนดให้มีรูปแบบการเรียกใช้งานดังนี้

```
fx = MyFx(x)
```

วิธีทำ โปรแกรมนี้สามารถสร้างได้ดังนี้

```
function fx = MyFx(x)

if (x >= 2) then
   fx = 2*x^2 - 3*cos(x);
elseif (x <= 2) then
   fx = sqrt(cosh(x)^2 + 1)
else
   fx = log10(x) + exp(x);
end

endfunction
```

เมื่อเขียนไฟล์ฟังก์ชันเสร็จก็ทำการบันทึกไฟล์นี้ในชื่อ MyFx.sci ตัวอย่างการใช้งานเช่น

```
-->getf('MyFx.sci')                    //ทำการโหลดไฟล์ฟังก์ชันก่อนเรียกใช้งาน

-->fx = MyFx(-3)
 fx  =
    10.117204

-->fx = MyFx(0.3)
 fx  =
    1.446628

-->fx = MyFx(2.2)
 fx  =
    11.445503
```

ตัวอย่างที่ 6 จงเขียนโปรแกรมเพื่อหาค่าของฟังก์ชัน $f(N)=\sum_{i=0}^{N}(i^2-3i)$ เมื่อ N เป็นเลขจำนวนเต็มบวก โดยกำหนดให้มีรูปแบบการเรียกใช้งานดังนี้

```
fn = MyFN(N)
```

วิธีทำ โปรแกรมนี้สามารถสร้างได้โดยใช้คำสั่ง for ดังนี้

```
function fn = MyFN(N)

fn = 0;
for i = 0:N
    fn = fn + (i^2 - 3*i);
end

endfunction
```

เมื่อเขียนไฟล์ฟังก์ชันเสร็จก็ทำการบันทึกไฟล์นี้ในชื่อ MyFN.sci ตัวอย่างการใช้งานเช่น

```
-->getf('MyFN.sci')                    //ทำการโหลดไฟล์ฟังก์ชันก่อนเรียกใช้งาน

-->fn = MyFN(3)
 fn  =
  - 4.
```

```
-->fn = MyFN(20)
 fn  =
    2240.
```

หรืออาจจะเขียนโปรแกรมนี้โดยใช้คำสั่ง while แทนคำสั่ง for ได้ดังนี้

```
function fn2 = MyFN2(N)
fn2 = 0;
i = 0;
while i <= N
      fn2 = fn2 + (i^2 - 3*i);
      i = i + 1;
end
endfunction
```

เมื่อเขียนไฟล์ฟังก์ชันเสร็จก็ทำการบันทึกไฟล์นี้ในชื่อ MyFN2.sci ตัวอย่างการใช้งานเช่น

```
-->getf('MyFN2.sci')                    //ทำการโหลดไฟล์ฟังก์ชันก่อนเรียกใช้งาน
-->fn2 = MyFN2(3)
 fn2  =
  - 4.
-->fn2 = MyFN2(20)
 fn2  =
    2240.
```

ซึ่งให้ผลลัพธ์เท่ากันกับโปรแกรมที่คำสั่ง for

5.7 สรุป

ในบทนี้ได้อธิบายถึงคำสั่งต่างๆ ที่ใช้เป็นองค์ประกอบในการพัฒนาโปรแกรม ได้แก่ คำสั่งวนซ้ำ (for และ while) ซึ่งเป็นตัวสั่งให้โปรแกรมทำซ้ำชุดคำสั่งเดิมภายในลูปเป็นจำนวนรอบตามที่กำหนด และคำสั่งทดสอบเงื่อนไข (if-then-end, if-then-else-end, if-then-

elseif-else-end, และ select-case) นอกจากนี้ยังได้อธิบายถึงหลักการพัฒนาฟังก์ชัน ข้อแตกต่างระหว่างไฟล์สคริปต์และไฟล์ฟังก์ชัน การสร้างไฟล์ไดอารี รวมทั้งให้ข้อแนะนำในการพัฒนาโปรแกรมให้มีประสิทธิภาพ และแสดงตัวอย่างการเขียนโปรแกรมเพื่อใช้ในการแก้ไขปัญหาโจทย์ทางคณิตศาสตร์ ดังนั้นจากที่อธิบายในบทนี้จะเห็นได้ว่าการพัฒนาฟังก์ชันรูปแบบใหม่ๆ ขึ้นมาใช้งานร่วมกับโปรแกรม SCILAB สามารถทำได้โดยง่าย

5.8 แบบฝึกหัดท้ายบท

5.1 กำหนดให้ตัวแปร N เป็นเลขจำนวนเต็มบวกใดๆ จงเขียนโปรแกรมโดยใช้คำสั่ง for เพื่อคำนวณหาค่าต่อไปนี้

5.1.1) $\sum_{i=1}^{N} i = 1+2+3+\ldots+N$

5.1.2) $\sum_{i=1}^{N} i^2 = 1^2+2^2+3^2+\ldots+N^2$

5.1.3) $\sum_{i=1}^{N} i^3 = 1^3+2^3+3^3+\ldots+N^3$

5.2 ทำเหมือนกับข้อ 5.1 โดยใช้คำสั่ง while แทน

5.3 จงอธิบายข้อแตกต่างระหว่างคำสั่ง for และคำสั่ง while

5.4 กำหนดให้ตัวแปร N เป็นเลขจำนวนเต็มบวกใดๆ จงพิสูจน์ว่า

5.4.1) $\sum_{i=1}^{N} i = \frac{N}{2}(N+1)$

5.4.2) $\sum_{i=1}^{N} i^2 = \frac{N}{6}(N+1)(2N+1)$

5.4.3) $\sum_{i=1}^{N} i^3 = \left(\frac{N}{2}(N+1)\right)^2$

5.4.4) $\sum_{i=1}^{N} \frac{1}{i(i+1)} = \frac{N}{N+1}$

5.5 จงเขียนโปรแกรมเพื่อคำนวณหาตัวคูณร่วมน้อย (least common multiple) ของเลขจำนวนเต็มบวกใดๆ พร้อมทั้งตรวจคำตอบได้ที่โดยใช้คำสั่ง `lcm`

5.6 จงเขียนโปรแกรมเพื่อคำนวณหาตัวหารร่วมมาก (greastest common divisor) ของเลขจำนวนเต็มบวกใดๆ พร้อมทั้งตรวจคำตอบได้ที่โดยใช้คำสั่ง `gcd`

5.7 จากสมการอนุกรมต่อไปนี้

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \ldots\right)$$

จงเขียนโปรแกรมเพื่อหาว่าจะต้องใช้สมการข้างต้นจำนวนกี่พจน์สำหรับหาค่าของ π เพื่อให้ได้ค่าที่ถูกต้องถึงทศนิยม 5 ตำแหน่ง นั่นคือ $\pi = 3.14159$

5.8 จงเขียนโปรแกรมสำหรับข้อความต่อไปนี้ โดยใช้คำสั่ง `if`

5.8.1) ถ้ารากที่สองของค่า x มีค่ามากกว่าค่า 0 ให้แสดงค่า x ออกทางหน้าต่างคำสั่ง

5.8.2) ถ้า $x \geq 0$ ให้แสดงค่า x ออกทางหน้าต่างคำสั่ง แต่ถ้า $x < 0$ ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “The value of x is less than 0”

5.8.3) ถ้า $x \geq 1$ ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “x = 1” แต่ถ้า $x \leq -1$ ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “x = –1” สำหรับค่า x อื่นๆ ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “x = 0”

5.8.4) ถ้า $x \geq 80$ ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “You got A” แต่ถ้า $70 \leq x < 80$ ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “You got B” แต่ถ้า $60 \leq x < 70$ ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “You got C” แต่ถ้า $x < 60$ ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “You got F”

5.9 จงเขียนโปรแกรมสำหรับข้อความต่อไปนี้ โดยใช้คำสั่ง `select-case`

5.9.1) ถ้า x = `'A'` (อินพุตอาร์กิวเมนต์เป็นตัวอักขระ) ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “x = A” ถ้า x = `'B'` ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “x = B” ถ้า x = `'C'` ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “x = C” สำหรับค่า x ค่าอื่นๆ ให้โปรแกรมหยุด

การทำงานแล้วแสดงผลออกทางหน้าต่างคำสั่งว่า “x must be A, B, or C only…. Please try again ”

5.9.2) ถ้า x = 4 ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “You got A” แต่ถ้า x = 3 ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “You got B” แต่ถ้า x = 2 ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “You got C” แต่ถ้า x = 1 ให้แสดงผลออกทางหน้าต่างคำสั่งว่า “You got F” สำหรับค่า x ค่าอื่นๆ ให้โปรแกรมหยุดการทำงานแล้วแสดงผลออกทางหน้าต่างคำสั่งว่า “x must be 1, 2, 3 or 4 only…. Please try again ”

5.10 จงเขียนไฟล์ฟังก์ชันที่มีรูปแบบการเรียกใช้งานคือ [a, b, c, d] = MoneyExchange(m) เพื่อทำหน้าที่เปลี่ยนจำนวนเงิน m บาท ให้เป็นธนบัตรใบละ 100 บาท จำนวน a ใบ, ธนบัตรใบละ 20 บาท จำนวน b ใบ, เหรียญสิบบาท จำนวน c เหรียญ และเหรียญหนึ่งบาท จำนวน d เหรียญ

5.11 จงเขียนโปรแกรมเพื่อหาค่าของฟังก์ชันต่อไปนี้

$$f(x)=\begin{cases}\sqrt{\left(x^2+2\right)} & ,-10\le x<-1\\ x^3-10 & ,-1\le x<5\\ \sin(x)-\cos(2x) & ,5\le x<10\\ 0 & \text{elsewhere}\end{cases}$$

5.12 จงเปรียบเทียบข้อแตกต่างระหว่างคำสั่ง `exec` และคำสั่ง `getf`

5.13 จงอธิบายข้อแตกต่างระหว่างไฟล์สคริปต์และไฟล์ฟังก์ชัน

5.14 จงอธิบายการใช้งานคำสั่ง `argn` พร้อมแสดงตัวอย่างการใช้งานคำสั่งนี้

5.15 จงเขียนฟังก์ชันแบบอินไลน์ (in-line function) ให้ทำงานเหมือนกับข้อ 5.4.1 และ 5.4.2 โดยใช้คำสั่ง `function-endfunction` และ `deff`

5.16 จงอธิบายการใช้งานไฟล์ไดอารี (file diary) พร้อมแสดงตัวอย่างการใช้งานคำสั่งนี้

5.17 จงอธิบายการใช้งานคำสั่ง `tic-toc` พร้อมแสดงตัวอย่างการใช้งานคำสั่งนี้

บทที่ 6

การวาดกราฟด้วย SCILAB

การทำงานในด้านต่างๆ หลังจากที่ทำการวิเคราะห์ข้อมูลเสร็จแล้ว ขั้นตอนต่อไปก็คือการนำเสนอผลงานวิจัยหรือผลการทดลองที่ได้มา ซึ่งโดยทั่วไปแล้วมักจะแสดงผลในรูปของรูปกราฟ (graph) เช่น แผนภูมิ กราฟแท่ง และกราฟเส้น เป็นต้น เนื่องจากสื่อความหมายให้เข้าใจได้โดยง่าย ดังนั้นในบทนี้จะขออธิบายถึงหลักการวาดกราฟโดยใช้โปรแกรม SCILAB ทั้งที่เป็นแบบสองมิติและสามมิติ

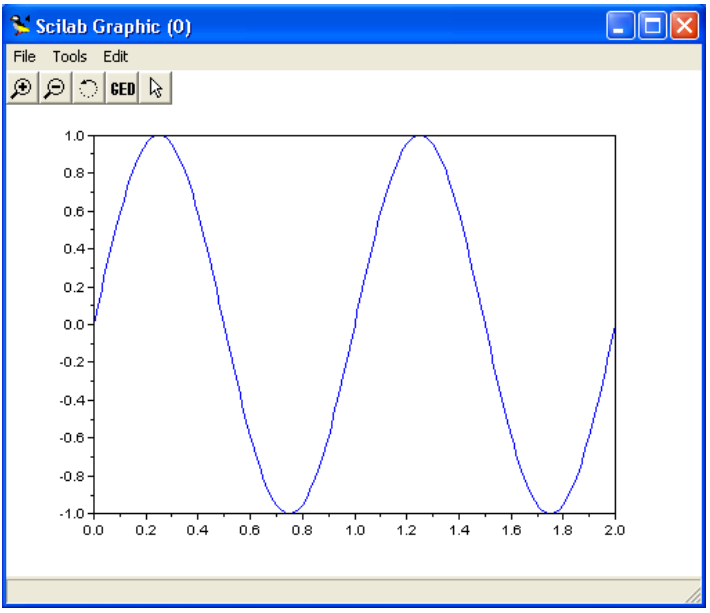
6.1 รายละเอียดหน้าต่างกราฟ

เมื่อใช้คำสั่งในการวาดกราฟ โปรแกรม SCILAB จะทำการสร้างหน้าต่างกราฟ (graphics window) ขึ้นมาใหม่ที่ชื่อว่า “SCILAB Graphic (j)” เพื่อแสดงผลของกราฟที่วาดขึ้นมา โดยที่ j เป็นเลขจำนวนเต็มที่มากกว่าหรือเท่ากับศูนย์ซึ่งแสดงถึงหมายเลขของหน้าต่างกราฟ ดังแสดงในรูปที่ 6.1 ซึ่งในที่นี้ j มีค่าเท่ากับค่า 0 จะหมายความว่าหน้าต่างที่ทำงานอยู่ (active window) ขณะนี้คือหน้าต่างกราฟหมายเลขศูนย์ จากรูปที่ 6.1 หน้าต่างกราฟจะมีแถบเมนู (menu bar) อยู่หนึ่งแถบประกอบไปด้วยสามเมนูหลัก คือ เมนู File, Tools และ Edit โดยที่แต่ละเมนูหลักจะมีเมนูย่อยตามที่แสดงในรูปที่ 6.2 ซึ่งรายละเอียดของแต่ละเมนูหลักสามารถอธิบายได้ดังต่อไปนี้

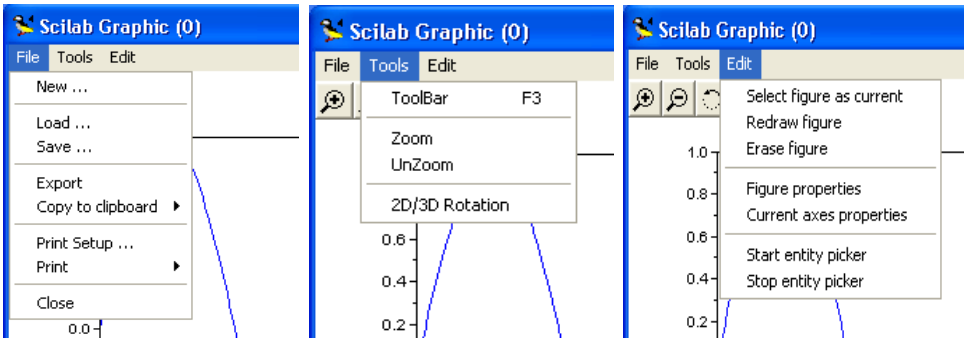
- เมนู **File** ประกอบไปด้วย

  New … สร้างหน้าต่างกราฟใหม่ขึ้นมา

  Load … เปิดรูปภาพที่บันทึกไว้ในไฟล์ตามรูปแบบ (format) ของโปรแกรม SCILAB (โดยทั่วไปจะเป็นไฟล์ที่ปิดท้ายด้วย `.scg`)



รูปที่ 6.1 ตัวอย่างหน้าต่างกราฟที่ใช้แสดงผลรูปภาพของโปรแกรม SCILAB



รูปที่ 6.2 รายละเอียดของเมนูหลักแต่ละอันในหน้าต่างกราฟ

Save …	บันทึกรูปกราฟลงไปเก็บไว้ในไฟล์ตามรูปแบบของโปรแกรม SCILAB (โดยทั่วไปจะเป็นไฟล์ที่ปิดท้ายด้วย `.scg`)
Export	บันทึกรูปกราฟลงไปในไฟล์ที่เป็นรูปแบบเฉพาะ (specific format) เช่น Postscript, Xfig, GIF, BMP, และ PPM เป็นต้น

Copy to clipboard	สำเนารูปกราฟเพื่อนำไปวาง (paste) ไว้ในที่อื่น
Print Setup …	กำหนดค่าพารามิเตอร์ต่างๆ ของเครื่องพิมพ์
Print	พิมพ์รูปกราฟออกทางเครื่องพิมพ์
Close	ปิดหน้าต่างกราฟที่ทำงานอยู่

- เมนู **Tools** ประกอบไปด้วย

ToolBar	เปิดหรือปิดแถบเครื่องมือที่ใช้ในการเรียกดูภาพหรือแก้ไขภาพ ตามที่แสดงในรูปของสัญรูป (icon) ทั้งห้าตัวบนมุมบนซ้ายของหน้าต่างกราฟ นั่นคือ 
Zoom	 ขยายขนาดของรูปกราฟเฉพาะส่วนที่ต้องการ สามารถทำได้โดยการใช้เมาส์ (mouse) เลื่อนเคอร์เซอร์ (cursor) ไปที่สัญรูปนี้ แล้วกดคลิกหนึ่งครั้ง จากนั้นก็ใช้เมาส์เลื่อนเคอร์เซอร์มายังตำแหน่งแรกที่ต้องการจะขยายรูปกราฟ กดคลิกหนึ่งครั้ง แล้วก็ใช้เมาส์เลื่อนเคอร์เซอร์เคอร์เซอร์ไปยังตำแหน่งสุดท้ายที่ต้องการจะขยายรูปกราฟ แล้วกดคลิกอีกหนึ่งครั้ง ก็จะได้รูปกราฟในส่วนที่ต้องการขยายตามที่ต้องการ
UnZoom	 ทำให้รูปกราฟที่ได้จากการขยาย กลับไปเป็นรูปกราฟเหมือนเดิมก่อนที่จะทำการขยาย
2D/3D Rotation	 หมุนรูปกราฟให้อยู่ในตำแหน่งและลักษณะที่ต้องการ
Graphical Editor	 เปิดหน้าต่างเอดิเตอร์เพื่อใช้ในการจัดรูปแบบการแสดงผลของรูปกราฟ เช่น กำหนดชื่อเส้นแกน x และเส้นแกน y, เปลี่ยนรูปแบบของเส้นกราฟ, และเปลี่ยนสีของกราฟ เป็นต้น
Entity Picker	 เปิดและปิดการเลือกตำแหน่งของรูปกราฟเพื่อทำการแก้ไข

- เมนู **Edit** ประกอบไปด้วย

Select figure as current	สำรองไว้ใช้ในโปรแกรม SCILAB เวอร์ชันถัดไป
Redraw figure	สำรองไว้ใช้ในโปรแกรม SCILAB เวอร์ชันถัดไป
Erase figure	ลบรูปกราฟที่แสดงอยู่ในหน้าต่างกราฟ

Figure properties	เปิดหน้าต่างเอดิเตอร์เพื่อใช้ในการจัดรูปแบบการแสดงผลของรูปกราฟเมนูย่อยนี้จะมีประโยชน์มาก โดยเฉพาะในกรณีที่ผู้ใช้ไม่สามารถจดจำค่าพารามิเตอร์ต่างๆ ที่ใช้ในคำสั่งสำหรับจัดรูปแบบการแสดงผลของรูปกราฟ
Current axes properties	เปิดหน้าต่างเอดิเตอร์เพื่อแสดงสถานะของเส้นแกน (axes) ที่ใช้อยู่ในหน้าต่างกราฟ
Start entity picker	เริ่มต้นการเลือกตำแหน่งของรูปกราฟเพื่อทำการแก้ไข
Stop entity picker	หยุดการเลือกตำแหน่งของรูปกราฟเพื่อทำการแก้ไข

6.2 การวาดกราฟสองมิติ

ในส่วนนี้จะอธิบายถึงพื้นฐานในการวาดกราฟสองมิติ และคำสั่งต่างๆ ที่ใช้ในการวาดกราฟสองมิติ พร้อมทั้งแสดงตัวอย่างการวาดกราฟสองมิติแบบต่างๆ

6.2.1 พื้นฐานการวาดกราฟสองมิติ

สมการคณิตศาสตร์แบบสองตัวแปรใดๆ สามารถที่จะแสดงให้อยู่ในรูปของกราฟสองมิติได้ เพื่อใช้แสดงความสัมพันธ์ของตัวแปรทั้งสอง การวาดกราฟสองมิติในโปรแกรม SCILAB นั้นไม่ยาก เพียงแต่จะต้องเข้าใจถึงรูปแบบของข้อมูลที่จะต้องป้อนให้กับคำสั่งวาดกราฟต่างๆ ก่อน คำสั่งพื้นฐานสำหรับการวาดกราฟสองมิติบนระบบพิกัดฉาก x-y คือคำสั่ง `plot` ซึ่งมีรูปแบบการเรียกใช้งาน ดังนี้

```
plot(x, y)
```

โดยตัวแปรหลักที่จะต้องป้อนให้กับคำสั่ง `plot` ประกอบด้วยเวกเตอร์สองชุด คือ เวกเตอร์ `x` ซึ่งเป็นตัวแปรอิสระที่กำหนดค่าในเส้นแกน `x` และเวกเตอร์ `y` ซึ่งเป็นตัวแปรตามที่กำหนดค่าในเส้นแกน `y` (โดยที่เวกเตอร์ `y` จะต้องมีขนาดเท่ากันกับเวกเตอร์ `x` เสมอ) นอกจากนี้คำสั่ง `plot` ยังสามารถที่จะถูกเรียกใช้งานได้ในอีกรูปแบบหนึ่ง คือ

```
plot(y)
```

ซึ่งในกรณีนี้โปรแกรม SCILAB จะสมมติว่าพารามิเตอร์ x มีค่าเท่ากับค่า 1 ถึงจำนวนสมาชิกทั้งหมดของเวกเตอร์ y นั่นคือ x = 1:length(y) โดยอัตโนมัติ

ตัวอย่างที่ 1 จงวาดกราฟของรูปสัญญาณไซนูซอยด์ (sinusoid waveform) ตามสมการ $y = \sin(2\pi ft)$ สำหรับเวลาที่ t = 0 ถึง 2 วินาที ถ้ากำหนดให้ความถี่ f = 1 เฮิตร์ซ (Hertz)

วิธีทำ จากโจทย์สามารถเขียนเป็นชุดคำสั่งของโปรแกรม SCILAB ได้ดังนี้

```
-->t = 0:0.01:2;
-->f = 1;
-->y = sin(2*%pi*f*t);
-->plot(t, y)
-->xtitle('A sinusoidal wave','Time (t)','Amplitude')
```

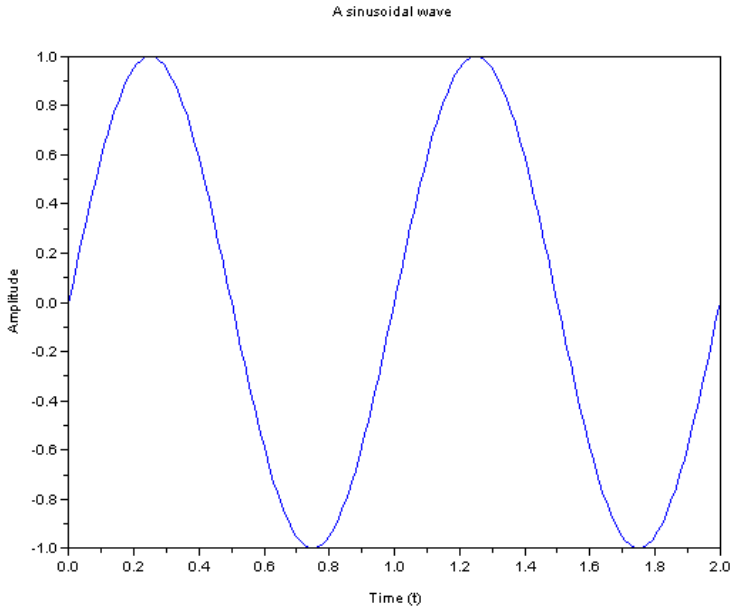
คำสั่งแรกเป็นการกำหนดให้ตัวแปร t ให้มีค่าอยู่ระหว่าง 0 ถึง 2 โดยที่สมาชิกแต่ละตัวที่อยู่ติดกันจะมีค่าห่างกันคงที่เท่ากับ 0.01 (ขนาดของตัวแปร t คือ 1×201) จากนั้นก็กำหนดค่าความถี่ f ให้เท่ากับหนึ่ง แล้วก็หาค่าของสัญญาณ y โดยค่า y ที่หามาได้จะมีขนาดเท่ากับตัวแปร t จากนั้นก็สั่งให้วาดกราฟขึ้นมาซึ่งผลลัพธ์ที่ได้จะเป็นกราฟตามรูปที่ 6.3 ส่วนคำสั่ง xtitle เป็นคำสั่งที่ใช้ในการกำหนดชื่อของกราฟ, ชื่อของเส้นแกน x, และชื่อของเส้นแกน y

---

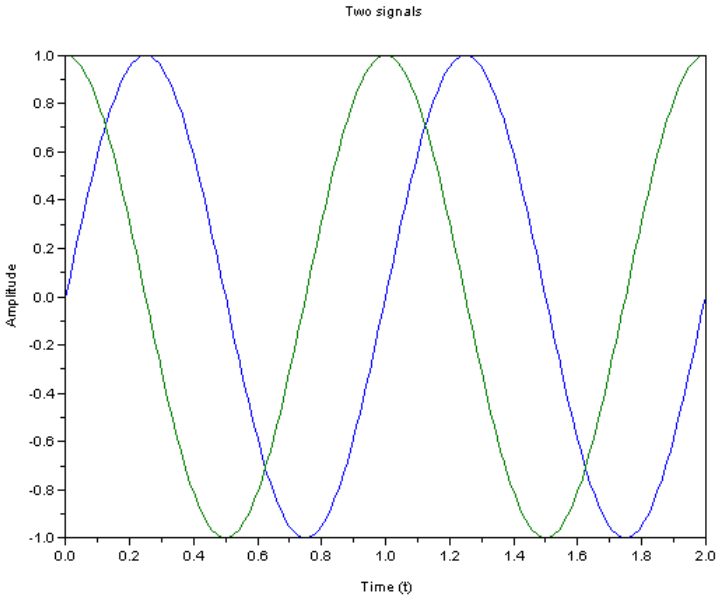
ในกรณีที่พารามิเตอร์ y เป็นเมทริกซ์ สมาชิกทั้งหมดในแต่ละแนวตั้งของเมทริกซ์ y จะถูกนำมาใช้วาดกราฟแต่ละเส้น ดังตัวอย่างต่อไปนี้

```
-->clf;  t = 0:0.01:2;                    //คำสั่ง clf ทำหน้าที่ลบรูปกราฟในหน้าต่างกราฟ
-->f = 1;
-->y = [sin(2*%pi*f*t)'  cos(2*%pi*f*t)'];
-->plot(t, y)                             //วาดกราฟสองเส้นพร้อมกัน
-->xtitle('Two sinusoidal waves', 'Time (t)', 'Amplitude')
```

โดยผลลัพธ์ที่ได้เป็นไปตามรูปที่ 6.4



รูปที่ 6.3 สัญญาณไซนูซอยด์ $y = \sin(2\pi ft)$



รูปที่ 6.4 สัญญาณ $\sin(2\pi ft)$ และ $\cos(2\pi ft)$

คำสั่งวาดกราฟสองมิติอีกรูปแบบหนึ่งที่น่าสนใจก็คือ

```
plot2di([x], y, [options])
```

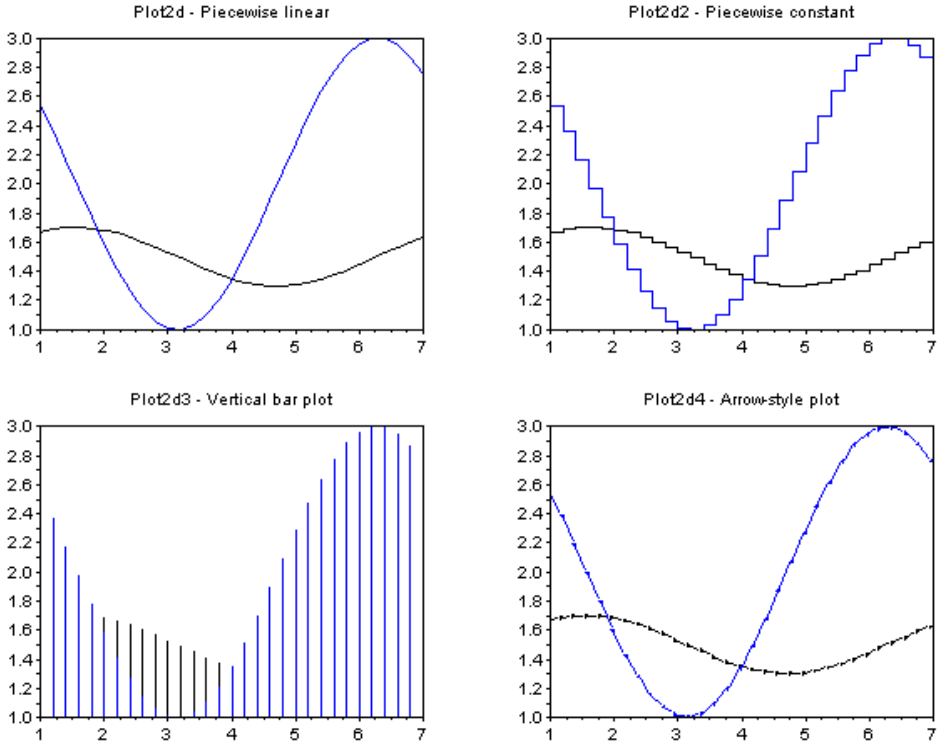
ซึ่งมีทั้งหมด 4 รูปแบบตามที่กำหนดโดยค่าพารามิเตอร์ i ดังนี้

1) ไม่มี i วาดกราฟเชิงเส้นโดยใช้สเกลแบบธรรมดาหรือสเกลแบบลอการิทึม (logarithm scale)
2) i = 2 วาดกราฟขั้นบันได (step function)
3) i = 3 วาดกราฟแท่ง (vertical bar)
4) i = 4 วาดกราฟเชิงเส้นโดยใช้ลูกศร (arrow) แสดงทิศทางของเส้นโค้ง (curve)

ตัวอย่างเช่น

```
-->clf;  t = (1:0.2:7)';

-->subplot(2, 2, 1);                                    //รูป 6.5 ด้านบนซ้าย
-->plot2d(t, [1.5+0.2*sin(t)  2+cos(t)]);
-->xtitle('Plot2d - Piecewise linear');

-->subplot(2, 2, 2);                                    //รูป 6.5 ด้านบนขวา
-->plot2d2(t, [1.5+0.2*sin(t)  2+cos(t)]);
-->xtitle('Plot2d2 - Piecewise constant');

-->subplot(2, 2, 3);                                    //รูป 6.5 ด้านล่างซ้าย
-->plot2d3(t, [1.5+0.2*sin(t)  2+cos(t)])
-->xtitle('Plot2d3 - Vertical bar plot')

-->subplot(2, 2, 4);                                    //รูป 6.5 ด้านล่างขวา
-->plot2d4(t, [1.5+0.2*sin(t)  2+cos(t)]);
-->xtitle('Plot2d4 - Arrow-style plot')
```

ซึ่งจะได้ผลลัพธ์ตามรูปที่ 6.5 จะเห็นได้ว่าชุดคำสั่งข้างต้นมีการเรียกใช้คำสั่ง subplot(m,n,i) เพื่อช่วยทำให้สามารถวาดกราฟได้หลายๆ กราฟในหน้าต่างกราฟเดียวกัน โดยคำสั่ง subplot



รูปที่ 6.5 ตัวอย่างรูปกราฟแบบต่างๆ ที่ได้จากการใช้คำสั่ง `plot2di`

จะทำหน้าที่ในการแบ่งหน้าต่างกราฟออกเป็นเมทริกซ์ขนาด `m×n` (`m` แถว และ `n` แนวตั้ง) สำหรับหน้าต่างกราฟย่อยแต่ละอัน โดยหน้าต่างกราฟย่อยอันที่ `i` (เมื่อ `i = 1, 2, ..., m*n`) จะเป็นหน้าต่างกราฟย่อยที่ถูกเรียกใช้งาน

พารามิเตอร์ `x` ในคำสั่ง `plot2di` คือ ตัวแปรอิสระที่กำหนดค่าในเส้นแกน `x` ซึ่งจะมีหรือไม่มีก็ได้ ในขณะที่พารามิเตอร์ `y` จะต้องอยู่ในรูปของเมทริกซ์ที่มีขนาด `nc×nl` เมื่อ `nc` คือจำนวนจุดที่จะวาดกราฟในแต่ละเส้นกราฟซึ่งจะต้องมีค่าเท่ากับจำนวนสมาชิกของพารามิเตอร์ `x` และ `nl` คือจำนวนเส้นกราฟที่จะวาด ในกรณีที่เป็นการวาดกราฟเพียงหนึ่งเส้นก็สามารถที่จะกำหนดให้พารามิเตอร์ `x` และ `y` เป็นเวกเตอร์แถวหรือเวกเตอร์แนวตั้งก็ได้ซึ่งจะให้ผลลัพธ์ออกมาเท่ากัน เช่น คำสั่ง `plot(t, cos(t))`, `plot(t', cos(t)')`, หรือ `plot(t, cos(t)')` จะให้ผลลัพธ์เหมือนกัน

ตารางที่ 6.1 การกำหนดลักษณะของจุดบนเส้นกราฟ

style = [j]	ลักษณะของจุด	คำอธิบาย
j = 0	.	จุด (point)
j = -1	+	บวก (plus)
j = -2	×	กากบาท (x-mark)
j = -3	⊕	วงกลมบวก (plus-circle)
j = -4	♦	สี่เหลี่ยมรูปเพชรแบบทึบ (black diamond)
j = -5	◇	สี่เหลี่ยมรูปเพชรแบบธรรมดา (diamond)
j = -6	Λ	สามเหลี่ยมชี้ขึ้น (triangle up)
j = -7	▽	สามเหลี่ยมชี้ลง (triangle down)
j = -8	⊕	สี่เหลี่ยมรูปเพชรบวก (plus-diamond)
j = -9	○	วงกลม (circle)

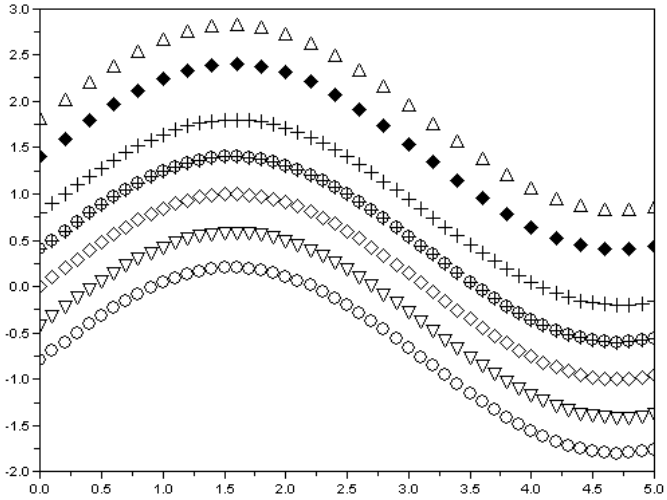
ส่วนพารามิเตอร์ options จะมีหรือไม่มีก็ได้ แต่ถ้ามีการเรียกใช้งาน options จะมีลักษณะการใช้งานดังนี้

```
options = [style, axesflag, leg, rect, logflag, nax, frameflag]
```

โดยที่

- style มีลักษณะการใช้งานคือ style = [j] เมื่อค่าของพารามิเตอร์ j จะเป็นตัวกำหนดรูปแบบของเส้นกราฟแบบต่างๆ ดังแสดงในตารางที่ 6.1 ตัวอย่างการใช้งานเช่น

```
-->x = 0:0.1:5;                                    //เวกเตอร์ x มีขนาด 1X51
-->u = [-0.8+sin(x); -0.4+sin(x); sin(x); ...
-->0.4+sin(x); 0.8+sin(x)]';                       //เมทริกซ์ u มีขนาด 51X5
-->plot2d(x, u, style=[-9, -7, -5, -3, -1])
-->x = 0:0.2:5;                                    //เวกเตอร์ x มีขนาด 1X26
-->y = [1.4+sin(x); 1.8+sin(x)]';                  //เมทริกซ์ y มีขนาด 26X2
-->plot2d(x, y, style=[-4, -6])
```



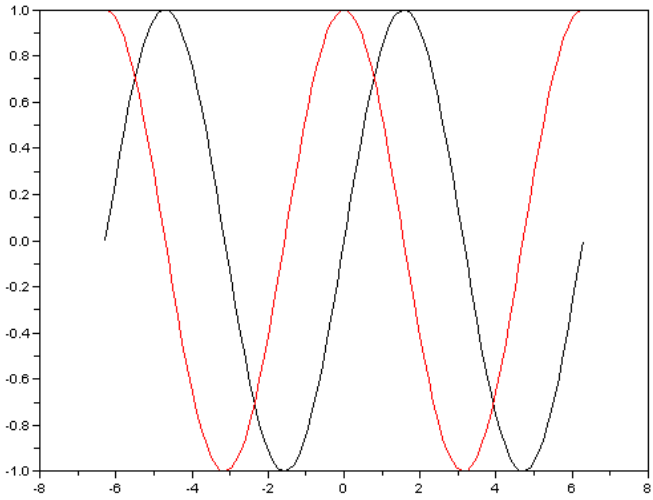
รูปที่ 6.6 ตัวอย่างรูปกราฟแสดงการใช้พารามิเตอร์ `style` ในคำสั่ง `plot2d`

ตารางที่ 6.2 ตัวอย่างการกำหนดลักษณะของสีเส้นกราฟ

`style = [j]`	สีของเส้นกราฟ
`j = 1`	สีดำ (black)
`j = 2`	สีน้ำเงิน (blue)
`j = 3`	สีเขียว (green)
`j = 4`	สีไซแอ็น (cyan) เขียวผสมน้ำเงิน
`j = 5`	สีแดง (red)
`j = 6`	สีม่วงแดงเข้ม (magenta)
`j = 7`	สีเหลือง (yellow)
`j = 8`	สีขาว (white)

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.6 แต่ถ้าพารามิเตอร์ `j` เป็นเลขจำนวนเต็มบวก ผลลัพธ์ที่ได้จากการใช้ `style = [j]` จะกลายเป็นการกำหนดสีของเส้นกราฟตามที่กำหนดในตารางที่ 6.2 ตัวอย่างเช่น

```
-->x = linspace(-2*%pi, 2*%pi, 100)';
-->plot2d(x, [sin(x),cos(x)], style=[1,5]);
```



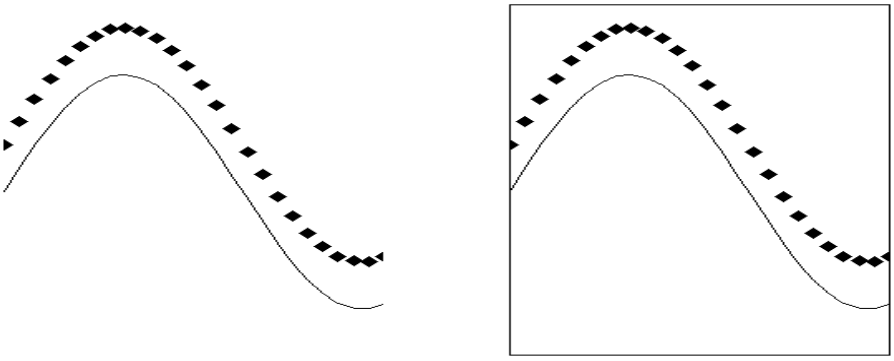
รูปที่ 6.7 ตัวอย่างรูปกราฟแสดงการใช้คำสั่งกำหนดสีของเส้นกราฟ

ตารางที่ 6.3 การกำหนดลักษณะของเส้นแกนในหน้าต่างกราฟ

`axesflag = [j]`	คำอธิบาย
`j = 0`	ไม่ต้องแสดงเส้นแกน (ทั้งเส้นแกน x และเส้นแกน y)
`j = 1`	แสดงเส้นแกนพร้อมข้อมูลของเส้นแกน โดยที่เส้นแกน y จะแสดงอยู่ทางด้านซ้าย (ค่าโดยปริยาย)
`j = 2`	แสดงเส้นแกน แต่ไม่ต้องแสดงข้อมูลของเส้นแกน
`j = 3`	แสดงเส้นแกนพร้อมข้อมูลของเส้นแกน โดยที่เส้นแกน y จะแสดงอยู่ทางด้านขวา
`j = 4`	แสดงเส้นแกน ณ ตำแหน่งกึ่งกลางของกรอบ (frame) รูปภาพ
`j = 5`	เป็นการกำหนดให้เส้นแกน x และเส้นแกน y จะต้องตัดกันที่จุด `(0,0)` ถ้าจุด `(0,0)` ไม่ได้อยู่ในกรอบรูปภาพ เส้นแกนจะไม่แสดงออกมา

โดยคำสั่ง `linspace(vmin, vmax, num)` จะเป็นการแบ่งค่าสูงสุด `vmax` กับค่าต่ำสุด `vmin` เป็นจำนวน `num` ตัว ผลลัพธ์ของชุดคำสั่งนี้แสดงในรูปที่ 6.7 ซึ่งจะได้กราฟสองเส้น เส้นหนึ่งมีสีดำ (สำหรับ `j = 1`) และอีกเส้นหนึ่งมีสีแดง (สำหรับ `j = 5`)

- `axesflag` มีลักษณะการใช้งานคือ `axesflag = [j]` โดยที่ค่าของพารามิเตอร์ `j` จะเป็นตัวกำหนดลักษณะของเส้นแกนในหน้าต่างกราฟโดยมีรูปแบบต่างๆ ตามที่แสดงในตารางที่ 6.3 ตัวอย่างการใช้งานพารามิเตอร์ `axesflag` มีดังนี้



รูปที่ 6.8 ตัวอย่างรูปกราฟแสดงการใช้พารามิเตอร์ `axesflag` ในคำสั่ง `plot2d`

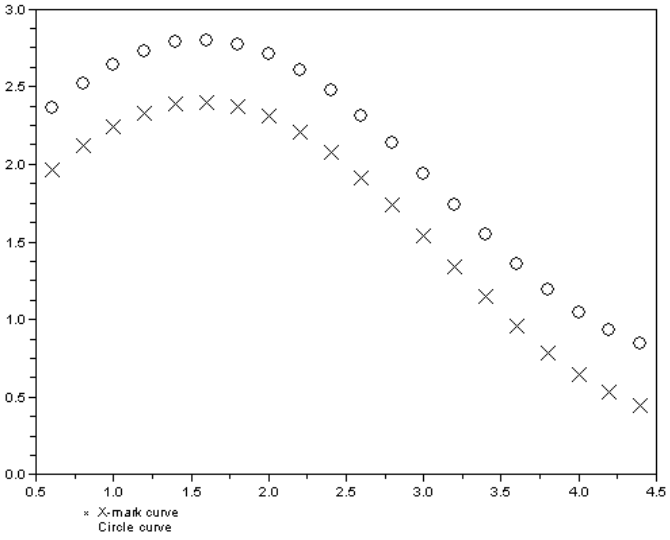
```
-->clf;  x = 0:0.2:5;
-->y = [1.4+sin(x); 1.8+sin(x)]';
-->subplot(1,2,1);  plot2d(x, y, style=[1,-4], axesflag=[0])
-->subplot(1,2,2);  plot2d(x, y, style=[1,-4], axesflag=[2])
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.8

- `leg` ใช้ในการใส่ข้อความบรรยาย (description) เส้นกราฟแต่ละเส้น ซึ่งมีลักษณะการใช้งาน คือ `leg = "description 1 @description 2 @ ..."` โดยที่ข้อความที่อยู่ภายในเครื่องหมาย `"..."` จะใช้บรรยายเส้นกราฟแต่ละเส้น และข้อความบรรยายเส้นกราฟแต่ละเส้นจะถูกคั่นด้วยเครื่องหมาย @
- `rect` ใช้กำหนดขนาดของกรอบรูปภาพซึ่งมีลักษณะการใช้งาน คือ `rect = [xmin, ymin, xmax, ymax]` โดยที่ `xmin` และ `xmax` ใช้กำหนดค่าต่ำสุดและค่าสูงสุดของเส้นแกน x ในขณะที่ `ymin` และ `ymax` ใช้กำหนดค่าต่ำสุดและค่าสูงสุดของเส้นแกน y ตามลำดับ สำหรับตัวอย่างการใช้งานพารามิเตอร์ `leg` และ `rect` มีดังนี้

```
-->clf;  x = 0:0.2:5;
-->y = [1.4+sin(x); 1.8+sin(x)]';
-->plot2d(x, y, style=[-2 -9], leg="X-mark curve@Circle ...
-->curve", rect=[0.5,0,4.5,3]);
```

ซึ่งผลลัพธ์ที่ได้แสดงในรูปที่ 6.9

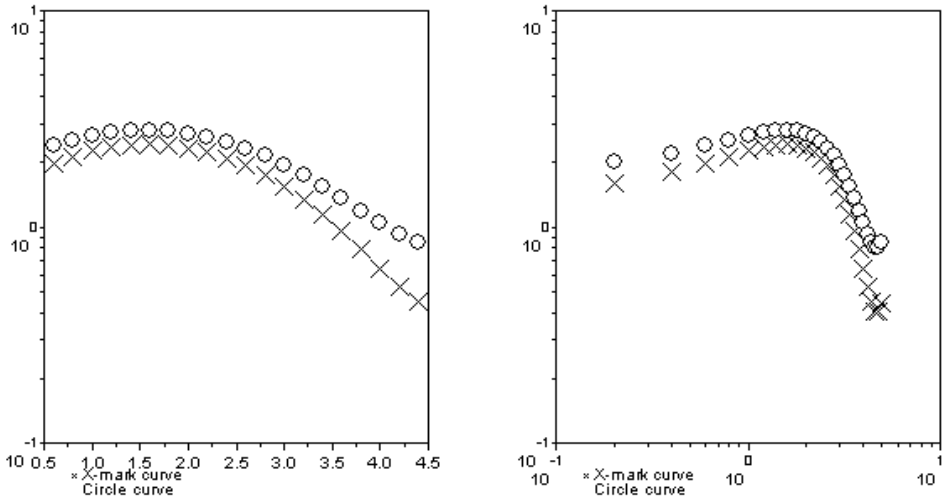


รูปที่ 6.9 ตัวอย่างรูปกราฟแสดงการใช้พารามิเตอร์ `leg` และ `rect` ในคำสั่ง `plot2d`

- `logflag` ใช้ในการกำหนดลักษณะของเส้นแกน `x` และเส้นแกน `y` ว่าจะให้เป็นสเกลแบบธรรมดาหรือสเกลแบบลอการิทึม (logarithm) โดยค่าที่เป็นได้ของ `logflag` มีทั้งหมดสี่แบบ คือ `"nn"`, `"nl"`, `"ln"` และ `"ll"` เมื่อ `n` หมายถึงให้ใช้สเกลแบบธรรมดา และ `l` หมายถึงให้ใช้สเกลแบบลอการิทึม (พารามิเตอร์ตัวแรกจะอ้างอิงถึงเส้นแกน `x` และพารามิเตอร์ตัวที่สองจะอ้างถึงเส้นแกน `y`) ตัวอย่างการใช้งานพารามิเตอร์ `logflag` มีดังนี้

```
-->clf;  x = 0:0.2:5;
-->y = [1.4+sin(x);  1.8+sin(x)]';
-->subplot(1, 2, 1);
-->plot2d(x, y, style=[-2 -9], leg="X-mark curve ...
-->@Circle curve", rect=[0.5,0.1,4.5, 3], logflag="nl");
-->subplot(1, 2, 2);
-->plot2d(x, y, style=[-2 -9], leg="X-mark curve ...
-->@Circle curve", rect=[0.5,0.1,4.5,3], logflag="ll");
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.10

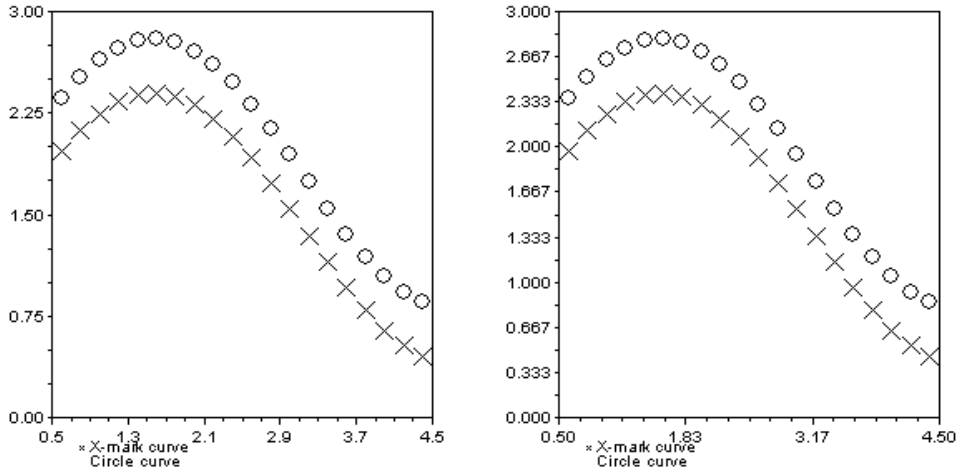


รูปที่ 6.10 ตัวอย่างรูปกราฟแสดงการใช้พารามิเตอร์ `logflag` ในคำสั่ง `plot2d`

- `nax` จะถูกนำมาใช้เมื่อมีการเรียกใช้พารามิเตอร์ `axesflag = 1` โดย `nax` จะใช้ในการกำหนดลักษณะของป้าย (label) ของเส้นแกน `x` และเส้นแกน `y` ซึ่งมีรูปแบบการใช้งานคือ `nax = [nx, Nx, ny, Ny]` โดยที่ `Nx` และ `Ny` เป็นตัวเลขจำนวนเต็มบวกที่ใช้กำหนดจำนวนขีดหลัก (main tics) ที่จะปรากฏบนเส้นแกน `x` และเส้นแกน `y` ในขณะที่ `nx` และ `ny` เป็นตัวเลขจำนวนเต็มบวกที่ใช้กำหนดจำนวนขีดย่อย (subtics) ที่จะปรากฏอยู่ระหว่างขีดหลักสองขีดบนเส้นแกน `x` และเส้นแกน `y` ตัวอย่างการใช้งานพารามิเตอร์ `nax` มีดังนี้

```
-->clf;  x = 0:0.2:5;
-->y = [1.4+sin(x);  1.8+sin(x)]';
-->subplot(1, 2, 1);
-->plot2d(x, y, style=[-2 -9], leg="X-mark curve @Circle ...
-->curve", rect=[0.5, 0, 4.5, 3], nax=[3, 6, 2, 5]);
-->subplot(1, 2, 2);
-->plot2d(x, y, style=[-2 -9], leg="X-mark curve @Circle ...
-->curve", rect=[0.5, 0,4. 5, 3], nax=[5, 4, 1, 10]);
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.11



รูปที่ 6.11 ตัวอย่างรูปกราฟแสดงการใช้พารามิเตอร์ `nax` ในคำสั่ง `plot2d`

- `frameflag` ใช้ควบคุมขอบเขตของเส้นแกน (axes boundaries) `x` และเส้นแกน `y` โดยมีลักษณะการใช้งาน คือ `frameflag = [j]` โดยที่ค่าของพารามิเตอร์ `j` จะเป็นตัวกำหนดลักษณะขอบเขตของเส้นแกนในหน้าต่างกราฟดังแสดงในตารางที่ 6.4

6.2.2 การตกแต่งรูปกราฟ

ในการนำเสนอผลงานวิจัยหรือผลการทดลองโดยใช้รูปกราฟ บางครั้งอาจจะมีความจำเป็นที่จะต้องตกแต่งรูปแบบของกราฟให้มีลักษณะต่างๆ เช่น เปลี่ยนสีและลักษณะของเส้นกราฟ, ใส่คำอธิบายลงไปในกราฟ, และอื่นๆ เพื่อให้อยู่ในรูปที่สวยงามและง่ายต่อความเข้าใจ การตกแต่งรูปกราฟในโปรแกรม SCILAB สามารถทำได้โดยใช้เมนูย่อย Figure properties (ในเมนูหลัก Edit ของหน้าต่างกราฟ) อย่างไรก็ตามในส่วนนี้จะขออธิบายถึงคำสั่งต่างๆ ที่ใช้บ่อยในการจัดรูปแบบการแสดงผลของกราฟ (ซึ่งมีประโยชน์ในกรณีที่ต้องการตกแต่งรูปกราฟอย่างง่ายโดยเร็ว) ดังต่อไปนี้

6.2.2.1 คำสั่ง `xtitle`

เป็นคำสั่งที่ใช้ในการกำหนดชื่อของรูปกราฟ, ชื่อของเส้นแกน `x`, และชื่อของเส้นแกน `y` โดยมีรูปแบบการใช้งาน คือ

```
xtitle('title_name','x_axis','y_axis')
```

ตารางที่ 6.4 การกำหนดลักษณะขอบเขตของเส้นแกนในหน้าต่างกราฟ

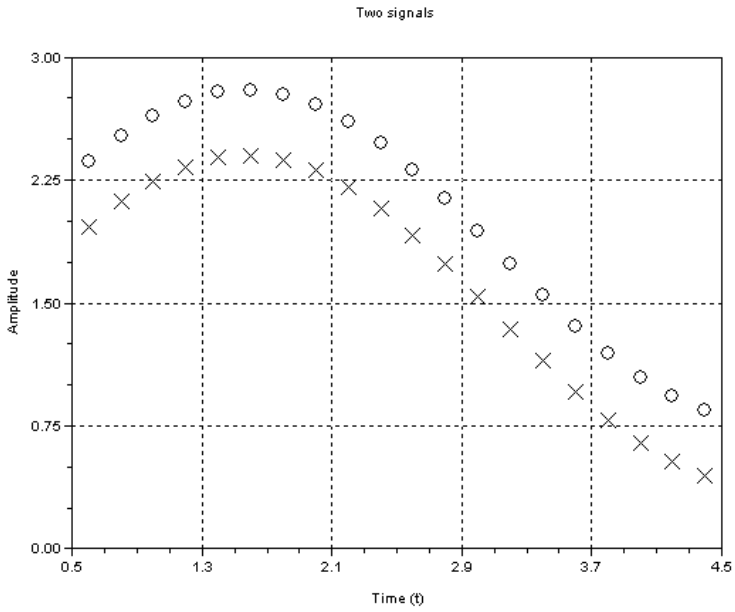
frameflag = [j]	คำอธิบาย
j = 0	ใช้สเกลเดิม (previous scale) ที่ใช้ในการวาดรูปกราฟก่อนหน้านี้ ในการวาดกราฟรูปใหม่ (เป็นค่าโดยปริยาย)
j = 1	ขอบเขต (range) ของหน้าต่างกราฟจะถูกกำหนดโดยพารามิเตอร์ rect
j = 2	ขอบเขตของหน้าต่างกราฟจะถูกกำหนดโดยค่าต่ำสุดและค่าสูงสุดของข้อมูลที่ใช้ในตัวแปร x และ y
j = 3	เหมือนกับ j = 1 แต่ใช้สเกลแบบสมมิติ (isometric scale)
j = 4	เหมือนกับ j = 2 แต่ใช้สเกลแบบสมมิติ
j = 5	เหมือนกับ j = 1 แต่อาณาเขตของหน้าต่างกราฟจะมีขนาดขึ้นเพื่อให้ป้ายของเส้นแกนสวยงาม (pretty axes)
j = 6	เหมือนกับ j = 2 แต่อาณาเขตของหน้าต่างกราฟจะมีขนาดขึ้นเพื่อให้ป้ายของเส้นแกนสวยงาม
j = 7	เหมือนกับ j = 5 แต่สเกลของกราฟใหม่จะไปทับซ้อนสเกลของเดิม
j = 8	เหมือนกับ j = 6 แต่สเกลของกราฟใหม่จะไปทับซ้อนสเกลของเดิม

เมื่อ title_name คือชื่อของรูปกราฟ, x_axis คือชื่อของเส้นแกน x, และ y_axis คือชื่อของเส้นแกน y

6.2.2.2 คำสั่ง xgrid(j)

เป็นคำสั่งที่ใช้ในการใส่เส้นกริดเข้าไปในรูปกราฟ โดยเส้นกริดจะมีสีตามที่กำหนดโดยพารามิเตอร์ j ซึ่งเป็นเลขจำนวนเต็มบวกตามที่แสดงในตารางที่ 6.2 ตัวอย่างการใช้งานคำสั่ง xgrid เช่น จากรูปที่ 6.11 ด้านซ้าย ถ้าต้องการเพิ่มเส้นกริดสีดำเข้าไปในรูปกราฟก็สามารถทำได้โดยใช้คำสั่งต่อไปนี้

```
-->clf;  x = 0:0.2:5;
-->y = [1.4+sin(x);  1.8+sin(x)]';
-->plot2d(x, y, style=[-2 -9], rect=[0.5,0,4.5,3], nax=[3,6,2,5]);
-->xtitle('Two signals','Time (t)','Amplitude')
-->xgrid(1)                              //ใส่เส้นกริดสีดำ (j = 1) เข้าไปในรูปกราฟ
```



รูปที่ 6.12 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `xgrid`

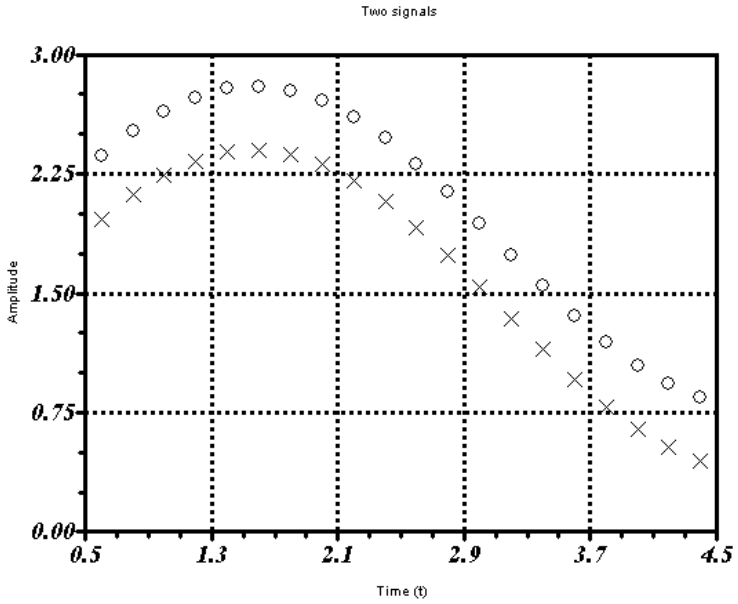
ซึ่งจะได้ผลลัพธ์ตามที่แสดงในรูปที่ 6.12

6.2.2.3 คำสั่ง `xset`

เป็นคำสั่งที่ใช้ในการปรับรูปแบบการแสดงผลของเส้นแกน `x` และเส้นแกน `y` คำสั่งนี้มีลักษณะการใช้งานหลายรูปแบบ เช่น เปลี่ยนรูปแบบและขนาดของตัวอักษรของเส้นแกน, เปลี่ยนความหนา (หรือความเข้ม) ของเส้นแกน, และอื่นๆ ตัวอย่างการใช้งานคำสั่งนี้ เช่น จากรูปที่ 6.12 ถ้าต้องการปรับรูปแบบของเส้นแกนก็สามารถทำได้โดยการเพิ่มคำสั่งต่อไปนี้ลงไป

```
-->xset("font", 5, 4)          //เพิ่มขนาดของตัวอักษรในเส้นแกน x และเส้นแกน y
-->xset("thickness", 3)        //เพิ่มขนาดของเส้นแกน x และเส้นแกน y
```

ซึ่งผลลัพธ์ที่ได้แสดงในรูปที่ 6.13 นอกจากนี้คำสั่ง `xset` ยังมีการใช้งานอีกหลายรูปแบบที่น่าสนใจ ผู้สนใจสามารถศึกษารายละเอียดการใช้งานของคำสั่ง `xset` เพิ่มเติมได้จากคำสั่ง `help`



รูปที่ 6.13 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `xset`

6.2.2.4 คำสั่ง `plotframe`

เป็นคำสั่งที่ใช้สร้างกรอบ (frame) สำหรับวาดรูปกราฟ โดยมีลักษณะการใช้งานดังนี้

```
plotframe(rect, tics, [options])
```

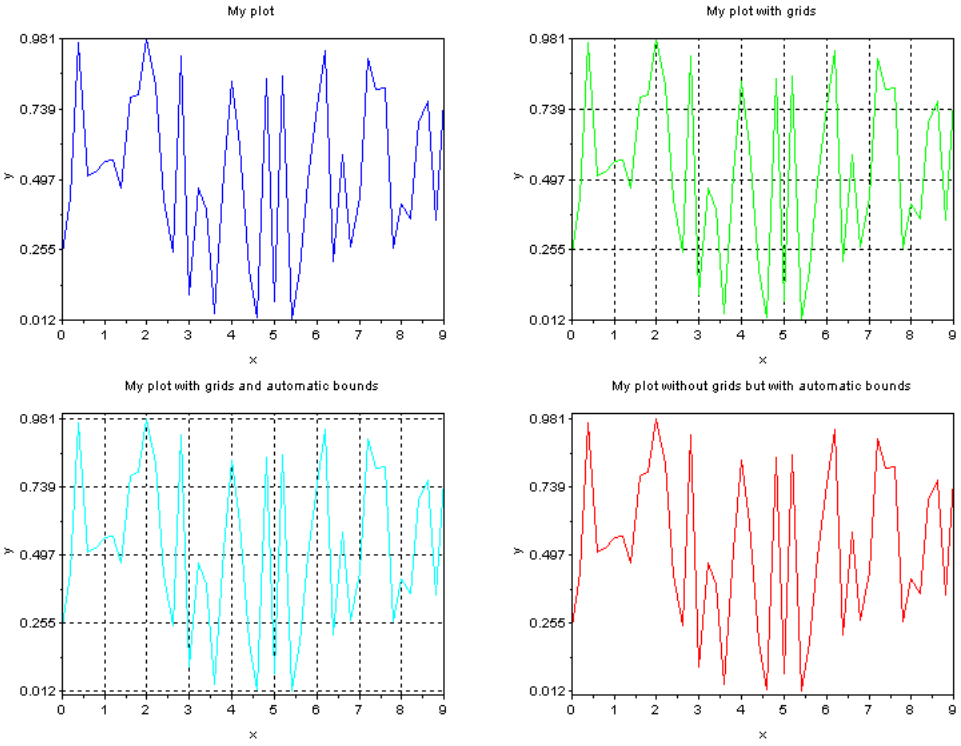
โดยที่ความหมายของพารามิเตอร์แต่ละตัว คือ

- `rect` เป็นเวกเตอร์ `[xmin, ymin, xmax, ymax]` ที่ใช้กำหนดค่าต่ำสุดและค่าสูงสุดของเส้นแกน `x` และ `y`
- `tics` เป็นเวกเตอร์ `[nx, Nx, ny, Ny]` โดยที่ `Nx` และ `Ny` คือจำนวนขีดหลัก (main tics) ที่จะแบ่งบนเส้นแกน `x` และ `y` ในขณะที่ `nx` และ `ny` คือจำนวนขีดย่อย (subtics) ที่จะแบ่งในแต่ละช่องของ `Nx` และ `Ny`
- `options` มีรูปแบบการใช้งานอยู่สามแบบ คือ

- flags เป็นเวกเตอร์ของตัวแปรบูลีน (%t หรือ %f) ขนาด 1×2 ประกอบไปด้วย [wantgrids, findbounds] โดยที่ wantgrids หมายถึงต้องการให้ใส่เส้นกริดเข้าไปในรูปกราฟหรือไม่ และ findbounds หมายถึงต้องการให้มีการปรับค่าขอบเขตของกรอบรูปภาพที่กำหนดใน rect โดยอัตโนมัติหรือไม่
- captions เป็นเวกเตอร์ขนาด 1×3 ประกอบด้วย [title, x-leg, y-leg] ใช้กำหนดชื่อเรื่อง, ชื่อเส้นแกน x, และชื่อเส้นแกน y ของรูปกราฟ โดยจะทำหน้าที่เทียบเท่ากับการใช้คำสั่ง xtitle
- subwins เป็นเวกเตอร์ขนาด 1×4 ประกอบด้วย [x, y, w, h] ใช้ในการกำหนดหน้าต่างกราฟย่อย (sub-window) โดยที่ พารามิเตอร์ x และ y แสดงตำแหน่งจุดพิกัดของหน้าต่างกราฟย่อย w แสดงขนาดความกว้าง (width) และ h แสดงขนาดความสูง (height) ของหน้าต่างกราฟย่อย

ตัวอย่างต่อไปนี้แสดงลักษณะการใช้งานของคำสั่ง plotframe

```
-->clf;  x = [0:0.2:9];
-->y = rand(x);
-->rect = [min(x), min(y), max(x), max(y)];
-->tics = [1, 10, 1, 5];
-->plotframe(rect, tics, [%f, %f], ["My plot", "x", "y"], ...
-->[0, 0, 0.5, 0.5])
-->plot2d(x, y, 2, "000")                    //รูปที่ 6.14 ด้านบนซ้าย
-->plotframe(rect, tics, [%t, %f], ["My plot with grids", ...
-->"x","y"],[0.5,0,0.5,0.5])
-->plot2d(x, y, 3, "000")                    //รูปที่ 6.14 ด้านบนขวา
-->plotframe(rect, tics, [%t, %t], ["My plot with grids and ...
-->automatic bounds", "x", "y"], [0, 0.5, 0.5, 0.5])
-->plot2d(x, y, 4, "000")                    //รูปที่ 6.14 ด้านล่างซ้าย
-->plotframe(rect, tics, [%f, %t], ["My plot without grids ...
-->but with automatic bounds", "x", "y"], [0.5, 0.5, 0.5, 0.5])
-->plot2d(x, y, 5, "000")                    //รูปที่ 6.14 ด้านล่างขวา
```



รูปที่ 6.14 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `plotframe`

โดยที่คำสั่ง `tics = [1, 10, 1, 5]` หมายถึงกำหนดให้เส้นแกน `x` มีสิบขีดหลักและมีหนึ่งขีดย่อยระหว่างขีดหลัก ส่วนเส้นแกน `y` มีห้าขีดหลักและมีหนึ่งขีดย่อยระหว่างขีดหลัก ผลลัพธ์ที่ได้จากชุดคำสั่งนี้แสดงในรูปที่ 6.14

6.2.2.5 คำสั่ง `xstring`

เป็นคำสั่งที่ใช้บรรจุสายอักขระเข้าไปในรูปกราฟตามรูปแบบและตำแหน่งที่กำหนด มีลักษณะการใช้งานดังนี้

```
xstring(x, y, str, angle, box)
```

โดยที่พารามิเตอร์

- x และ y เป็นเลขจำนวนจริงที่ใช้บอกจุดพิกัด (coordinate point) เริ่มต้นของสายอักขระ โดยจุดพิกัดเริ่มต้น x = 0 และ y = 0 จะอยู่ที่มุมล่างด้านซ้ายของกรอบรูปกราฟ
- str เป็นเมทริกซ์ของสายอักขระที่จะปรากฏในรูปกราฟตามจุดพิกัดที่กำหนด
- angel เป็นเลขจำนวนจริงที่ใช้แสดงมุมองศา (ระดับความเอียง) ของตัวสายอักขระในทิศทางตามเข็มนาฬิกา (ค่าโดยปริยายคือ ค่า 0)
- box เป็นเลขจำนวนเต็มที่ใช้ในการสร้างกล่องสี่เหลี่ยม (box) มาครอบสายอักขระ (ค่าโดยปริยายคือ ค่า 0 ซึ่งหมายถึงไม่ต้องมีกล่องสี่เหลี่ยมมาครอบสายอักขระ)

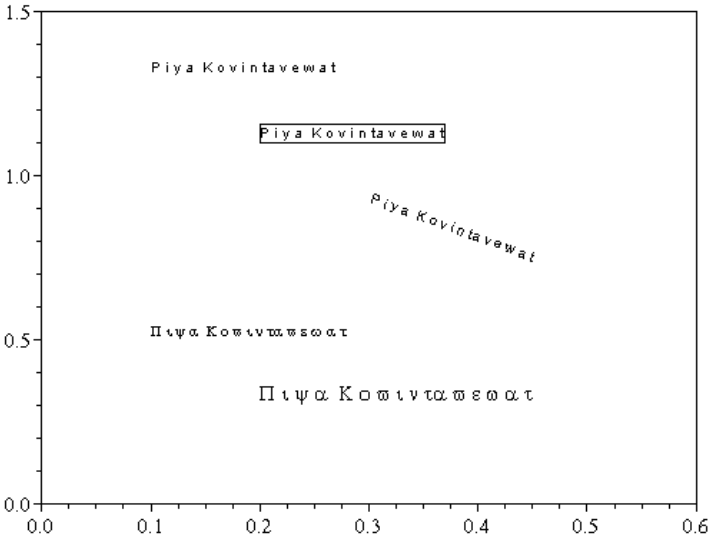
ตัวอย่างการใช้คำสั่ง xstring มีดังนี้

```
-->clf;
-->alphabet = ["P i y a  K o v i n ta v e w a t"];
-->plot2d([0;0.6], [0;1.5], 0);    //สร้างรูปกราฟขึ้นมา โดยไม่ต้องให้แสดงเส้นกราฟ
-->xstring(0.1, 1.3, alphabet)     //แสดงชื่อ "Piya" ที่จุดพิกัด x = 0.1 และ y = 1.8
-->xstring(0.2, 1.1, alphabet, 0, 1)      //แสดงชื่อ "Piya" ในกล่อง
-->xstring(0.3, 0.9, alphabet, 20)        //ให้ชื่อที่แสดงออกมามีมุมเอียงเท่ากับ 20°
-->xset("font", 1, 1)                     //กำหนดให้ใช้ตัวอักษรแบบสัญลักษณ์
-->xstring(0.1, 0.5, alphabet)
-->xset("font", 1, 3)                     //เปลี่ยนขนาดของตัวอักษร
-->xstring(0.2, 0.3, alphabet)
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.15

6.2.2.6 คำสั่ง scf(j)

เป็นคำสั่งที่ใช้กำหนดหน้าต่างกราฟหมายเลข j ให้เป็นหน้าต่างกราฟที่จะทำงาน (active graphic window) โดยที่ j เป็นเลขจำนวนเต็มบวก แต่ถ้าหน้าต่างหมายเลขนั้นยังไม่มีหรือไม่มีการกำหนดค่าพารามิเตอร์ j) โปรแกรม SCILAB ก็จะสร้างหน้าต่างกราฟหมายเลข j ขึ้นมาใหม่ คำสั่ง scf(j) จะทำหน้าที่เทียบเท่ากับการใช้คำสั่ง xset("window", j)



รูปที่ 6.15 รูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `xstring`

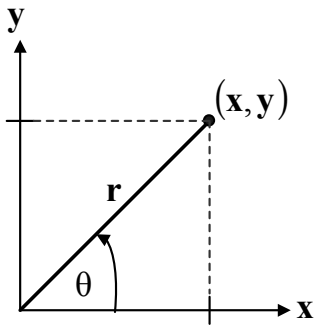
6.2.2.7 คำสั่ง `clf(j)`

เป็นคำสั่งที่มาจากคำว่า “Clear current graphic figure” โดยจะทำหน้าที่ลบรูปกราฟที่แสดงอยู่ในหน้าต่างกราฟหมายเลข `j` ในกรณีที่ไม่มีการกำหนดค่าพารามิเตอร์ `j` โปรแกรม SCILAB จะทำการลบรูปกราฟที่แสดงอยู่ในหน้าต่างกราฟที่กำลังทำงานอยู่

ถ้าไม่ใช้คำสั่ง `clf(j)` ก่อนที่จะสั่งให้โปรแกรม SCILAB วาดรูปกราฟใหม่ลงไปในหน้าต่างกราฟหมายเลข `j` ผลลัพธ์ที่ได้คือ รูปกราฟใหม่ที่สั่งให้วาดจะไปทับซ้อนกับรูปกราฟเดิมที่แสดงอยู่ในหน้าต่างกราฟหมายเลข `j`

6.2.2.8 คำสั่งอื่นๆ

- คำสั่ง `xbasc` ใช้ลบรูปกราฟที่แสดงอยู่ในหน้าต่างกราฟ คำสั่งนี้มีผลลัพธ์เทียบเท่ากับการใช้คำสั่ง `clf`
- คำสั่ง `xdel(j)` ใช้ลบหน้าต่างกราฟหมายเลข `j` ในกรณีที่ไม่มีการกำหนดค่า `j` โปรแกรม SCILAB ก็จะทำการลบหน้าต่างกราฟที่กำลังทำงานอยู่



รูปที่ 6.16 ความสัมพันธ์ระหว่างจุด (x, y) ในระบบพิกัดฉาก และจุด (r, θ) ในระบบพิกัดเชิงขั้ว

- คำสั่ง `xget` ใช้เรียกดูค่าพารามิเตอร์ต่างๆ ที่ใช้ในการกำหนดรูปแบบของรูปกราฟ กล่าวคือค่าพารามิเตอร์ทั้งหมดที่ใช้ในคำสั่ง `xset` สามารถที่จะถูกเรียกออกมาดูได้โดยใช้คำสั่ง `xget`
- คำสั่ง `xgetech` ใช้เรียกดูค่าพารามิเตอร์ต่างๆ ที่ใช้ในการกำหนดขนาดหรือสเกลของรูปกราฟ

6.2.3 กราฟเชิงขั้ว

นอกจากการวาดกราฟสองมิติแบบทั่วไปในระบบพิกัดฉาก `x-y` ตามที่อธิบายไว้ในหัวข้อที่ผ่านมาแล้ว โปรแกรม SCILAB ยังสามารถที่จะวาดกราฟในระบบพิกัดเชิงขั้ว (polar coordinates) ได้ดังแสดงต่อไปนี้

โดยทั่วไปจุด (x, y) ที่แสดงถึงตำแหน่ง (location) บนรูปกราฟในระบบพิกัดฉากสามารถที่จะเปลี่ยนให้อยู่ในรูปของจุด (r, θ) ในระบบพิกัดเชิงขั้วได้ โดยที่ r คือขนาด และ θ คือมุมเรเดียน (เทียบกับแกน x ในทิศทวนเข็มนาฬิกา) โดยอาศัยกฎของตรีโกณมิติ นั่นคือจากรูปที่ 6.16 จะได้ว่า

$$r = \sqrt{x^2 + y^2} \quad \text{และ} \quad \theta = \tan^{-1}\left(\frac{y}{x}\right)$$

ในทำนองเดียวกันจุดพิกัด (r, θ) ในระบบพิกัดเชิงขั้วก็สามารถที่จะแปลงกลับไปเป็นจุดพิกัด (x, y) ในระบบพิกัดฉากได้จากความสัมพันธ์ดังนี้

$$x = r\cos(\theta) \quad \text{และ} \quad y = r\sin(\theta)$$

การวาดกราฟเชิงขั้วในโปรแกรม SCILAB สามารถทำได้โดยการใช้คำสั่ง

```
polarplot(theta, r, [options])
```

เมื่อพารามิเตอร์ `theta` คือค่ามุม θ (มีหน่วยเป็นเรเดียน), พารามิเตอร์ `r` คือค่าความยาวของรัศมี, และพารามิเตอร์ `options` มีรูปแบบการใช้งานคือ

```
options = [style, strf, leg, rect]
```

โดยที่

- `style`, `leg`, และ `rect` มีลักษณะการใช้งานตามที่ได้กล่าวไว้แล้วในหัวข้อที่ผ่านมา
- `strf = "xy0"` (ค่าโดยปริยายคือ `strf = "030"`) โดยที่
  - `x` เป็นตัวควบคุมการแสดงผลของคำอธิบายในรูปกราฟ (caption) มีค่าดังนี้
    `x = 0` ไม่ต้องมีคำอธิบายในรูปกราฟ
    `x = 1` แสดงคำอธิบายในรูปกราฟ ที่กำหนดโดยพารามิเตอร์ `leg`
  - `y` มีลักษณะการใช้งานเหมือนพารามิเตอร์ `frameflag` ตามที่ได้กล่าวไว้แล้วในหัวข้อที่ผ่านมา

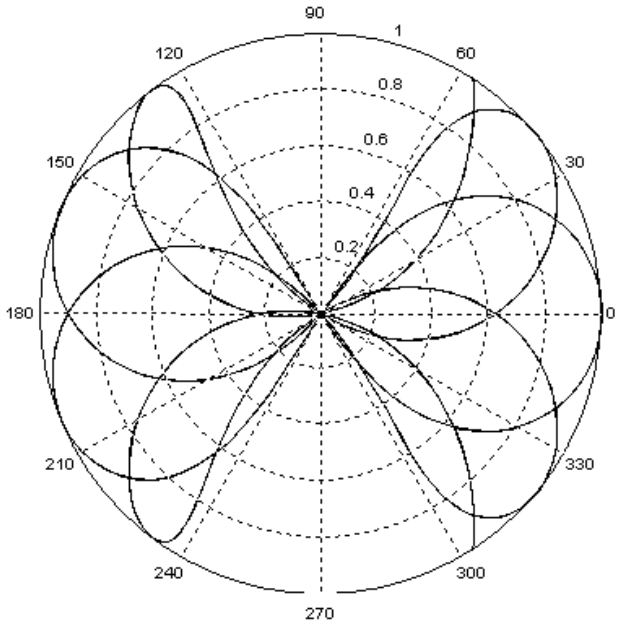
ตัวอย่างการใช้งานคำสั่ง `polarplot` เช่น

```
-->clf;  t = 0:0.01:2*%pi;
-->polarplot(sin(7*t), cos(8*t))
```

ผลลัพธ์แสดงในรูปที่ 6.17

6.2.4 การวาดกราฟสองมิติแบบพิเศษ

นอกจากนี้โปรแกรม SCILAB ยังได้เตรียมคำสั่งสำหรับการวาดกราฟสองมิติแบบอื่นๆ ไว้ใช้งานเฉพาะด้านมากมายดังแสดงในตารางที่ 6.5 (ผู้สนใจสามารถศึกษารายละเอียดเพิ่มเติมของคำสั่งวาดกราฟสองมิติเหล่านี้ได้จากคำสั่ง `help`) ตัวอย่างการใช้งานคำสั่งเหล่านี้ เช่น



รูปที่ 6.17 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `polarplot`

ตารางที่ 6.5 ตัวอย่างคำสั่งในการวาดกราฟสองมิติสำหรับการใช้งานเฉพาะด้าน

คำสั่ง	คำอธิบาย
`fplot2d`	วาดกราฟทั่วไปแบบสองมิติ ที่กำหนดโดยฟังก์ชันทางคณิตศาสตร์
`grayplot`	วาดกราฟสีแสดงพื้นผิว (surface) แบบสองมิติ
`fgrayplot`	วาดกราฟสีแสดงพื้นผิวแบบสองมิติ ที่กำหนดโดยฟังก์ชันทางคณิตศาสตร์
`histplot`	วาดกราฟฮีสโตแกรม (histogram plot)
`contour2d`	วาดกราฟคอนทัวร์ (contour surface) จากรูปกราฟสองมิติ
`champ`	วาดกราฟสนามเวกเตอร์แบบสองมิติ (2-D vector field)
`fchamp`	วาดกราฟสนามเวกเตอร์แบบสองมิติ ที่กำหนดโดยสมการอนุพันธ์อันดับหนึ่ง (first-order ordinary differential equation)
`bode`	วาดกราฟของโบดไดอะแกรม (Bode diagram) ทั้งกราฟแสดงขนาด (magnitude plot) และกราฟแสดงมุม (phase plot) ซึ่งมีประโยชน์มากทางด้านวิศวกรรม
`gainplot`	วาดกราฟแสดงขนาดของโบดไดอะแกรม
`nyquist`	วาดกราฟไนควิสต์ (Nyquist plot)
`evans`	วาดกราฟอีแวนรูทโลคัส (Evans root locus)
`plzr`	วาดกราฟโพล-ซีโร่ (pole-zero plot)

```
-->subplot(2, 2, 1);
-->champ(-5:5, -5:5, rand(11,11), rand(11,11), ...
-->rect = [-10,-10,10,10], arfact = 2);          //รูปที่ 6.18 ด้านบนซ้าย
-->xtitle('2-D vector field plot');
-->s = poly(0, 's');                             //กำหนดให้ตัวแปร s เป็นตัวแปรพหุนาม
-->h = syslin('c', (s^2 + 2*0.9*10*s + 100)/(s^2 + ...
-->2*0.3*10.1*s + 102.01));                       //กำหนดรูปแบบฟังก์ชันของระบบควบคุม
-->subplot(2, 2, 2);  nyquist(h, 0.01, 100);     //รูปที่ 6.18 ด้านบนขวา
-->subplot(2, 2, 3);  evans(h, 100);             //รูปที่ 6.18 ด้านล่างซ้าย
-->subplot(2, 2, 4);  plzr(h);                   //รูปที่ 6.18 ด้านล่างขวา
```

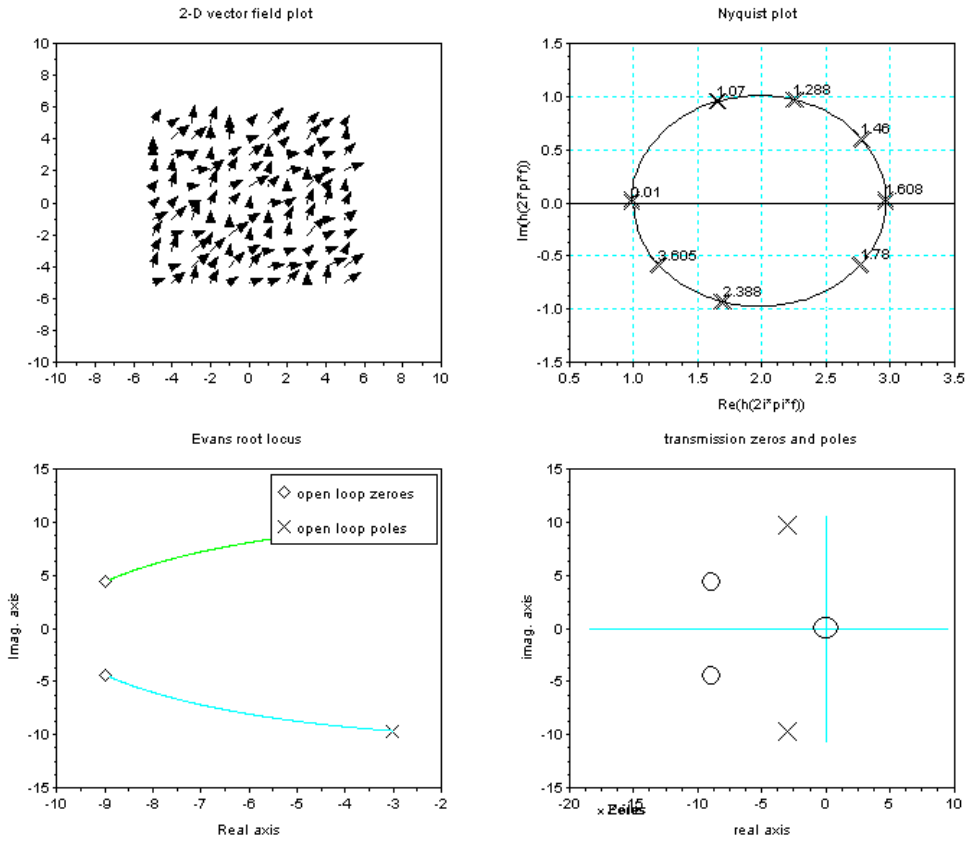
ผลลัพธ์ที่ได้จากชุดคำสั่งนี้แสดงในรูปที่ 6.18

6.3 การวาดกราฟสามมิติ

นอกจากการวาดกราฟสองมิติแล้ว โปรแกรม SCILAB ยังได้เตรียมคำสั่งจำนวนมากสำหรับการวาดกราฟสามมิติ ดังนั้นในส่วนนี้จะอธิบายถึงพื้นฐานในการวาดกราฟสามมิติ คำสั่งที่ใช้ในการวาดกราฟสามมิติ พร้อมทั้งแสดงตัวอย่างการวาดกราฟสามมิติแบบต่างๆ

6.3.1 พื้นฐานการวาดกราฟสามมิติ

สมการคณิตศาสตร์แบบสามตัวแปรใดๆ สามารถที่จะแสดงให้อยู่ในรูปของกราฟสามมิติได้ เพื่อใช้แสดงความสัมพันธ์ของตัวแปรทั้งสาม การใช้งานคำสั่งวาดกราฟสามมิตินั้นไม่ยากเพียงแต่ต้องเข้าใจถึงรูปแบบของข้อมูลที่จะป้อนให้กับคำสั่งเหล่านี้ การวาดกราฟสามมิติจะใช้ข้อมูลทั้งหมดสามชุดสำหรับเส้นแกน x, เส้นแกน y, และเส้นแกน z ที่อยู่ในพิกัดคาร์ทีเซียน (Cartesian coordinate) x-y-z โดยเวกเตอร์ x จะเป็นตัวกำหนดค่าในเส้นแกน x, เวกเตอร์ y จะเป็นตัวกำหนดค่าในเส้นแกน y, และตัวแปรตามที่มีค่าเปลี่ยนแปลงไปตามค่า x และ y ซึ่งก็คือขนาดของค่าบนเส้นแกน z นั่นเอง ดังนั้นตัวแปรตาม z นี้จะต้องมีจำนวนเท่ากับผลคูณของจำนวนข้อมูลในเวกเตอร์ x กับจำนวนข้อมูลในเวกเตอร์ y



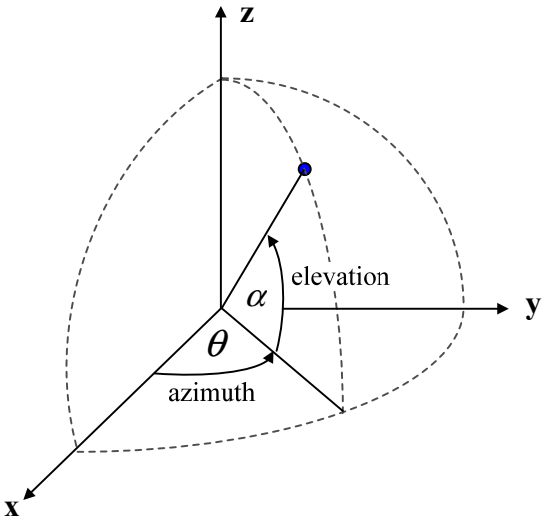
รูปที่ 6.18 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้ชุดคำสั่งสำหรับวาดกราฟสองมิติที่ใช้งานเฉพาะด้าน

คำสั่งพื้นฐานสำหรับการวาดกราฟแบบสามมิติในโปรแกรม SCILAB มีรูปแบบดังนี้

```
plot3d(x, y, z, [options])
```

โดยที่พารามิเตอร์ x และ y คือเวกเตอร์ที่มีขนาดเท่ากัน และพารามิเตอร์ z คือตัวแปรตามที่ขึ้นกับค่าของ x และ y ซึ่งจะมีจำนวนเท่ากับผลคูณของจำนวนข้อมูลในเวกเตอร์ x กับจำนวนข้อมูลในเวกเตอร์ y สำหรับพารามิเตอร์ options มีลักษณะการใช้งานคือ

```
options = [alpha, theta, leg, flag, ebox]
```



รูปที่ 6.19 รูปกราฟแสดงมุมยกและมุมกวาดในพิกัดทรงกลม

เมื่อพารามิเตอร์

- `alpha` และ `theta` คือเลขจำนวนจริงที่ใช้ในการกำหนดมุมองศาของจุดที่กำลังถูกพิจารณา (observation point) ในระบบพิกัดทรงกลม (spherical coordinate) ตามแสดงในรูปที่ 6.19 เมื่อ
  - `alpha` แสดงมุมยก (elevation angle) โดยค่า `alpha` ที่เป็นบวก หมายถึงมุมที่ยกขึ้น และค่า `alpha` ที่เป็นลบ หมายถึงมุมที่ยกลง (ค่าโดยปริยายคือ `alpha = 35`)
  - `theta` แสดงมุมกวาด (azimuth angle) โดยค่า `theta` ที่เป็นบวก หมายถึงมุมที่ทวนเข็มนาฬิกา และค่า `theta` ที่เป็นลบ หมายถึงมุมที่ตามเข็มนาฬิกา (ค่าโดยปริยายคือ `theta = 45`)
- `leg` เป็นพารามิเตอร์ที่ใช้ในการใส่ข้อความอธิบายเส้นกราฟแต่ละเส้น ตามที่กล่าวไว้แล้วในหัวข้อที่ผ่านมา
- `flag` เป็นเวกเตอร์ที่ประกอบไปด้วยพารามิเตอร์สามตัว คือ

```
flag = [mode, type, box]
```

โดยที่

- o mode เป็นเลขจำนวนเต็มที่ใช้กำหนดสีของพื้นผิวในรูปกราฟ เมื่อ
  - mode > 0 พื้นผิวในรูปกราฟจะเป็นสีตามหมายเลขของ mode ตามตารางที่ 6.2
  - mode = 0 พื้นผิวในรูปกราฟจะไม่มีสีหรือโปร่งแสง (transparent)
  - mode < 0 พื้นผิวในรูปกราฟจะเป็นสีตามหมายเลขของ |mode| แต่เส้นแสดงขอบเขตของรูปกราฟจะไม่ถูกแสดง
- o type เป็นเลขจำนวนเต็มที่ใช้กำหนดสเกลของรูปกราฟ เมื่อ
  - type = 0 สเกลที่ใช้วาดกราฟสามมิติเป็นแบบธรรมดา
  - type = 1 สเกลแบบอัตโนมัติ โดยเส้นแสดงขอบเขตจะถูกกำหนดโดยค่าพารามิเตอร์ ebox
  - type = 2 สเกลแบบอัตโนมัติ โดยเส้นแสดงขอบเขตจะถูกกำหนดจากข้อมูลที่ใช้ในการวาดกราฟ (เป็นค่าโดยปริยาย)
  - type = 3 สเกลแบบสมมิติ โดยเส้นแสดงขอบเขตจะถูกกำหนดโดยค่าพารามิเตอร์ ebox (คล้าย type = 1)
  - type=4 สเกลแบบสมมิติ โดยเส้นแสดงขอบเขตจะถูกกำหนดจากข้อมูลที่ใช้ในการวาดกราฟ (คล้าย type = 2)
  - type = 5 ความหมายเหมือนกับ type = 3 แต่จะมีขนาดใหญ่กว่า
  - type = 6 ความหมายเหมือนกับ type = 4 แต่จะมีขนาดใหญ่กว่า
- o box เป็นเลขจำนวนเต็มที่ใช้กำหนดลักษณะของกรอบรูปภาพ เมื่อ
  - box = 0 ไม่ต้องแสดงเส้นแกนของรูปกราฟ
  - box = 1 เหมือนกับ box = 0
  - box = 2 เฉพาะเส้นแกนที่อยู่หลังพื้นผิวของรูปกราฟจะถูกแสดงออกมา
  - box = 3 มีการสร้างกล่องขึ้นมาครอบรูปกราฟ พร้อมทั้งใส่ชื่อของเส้นแกน (x, y, และ z) ลงไป
  - box = 4 มีการสร้างกล่องขึ้นมาครอบรูปกราฟ พร้อมทั้งใส่ชื่อและสเกลของเส้นแกนลงไป (เป็นค่าโดยปริยาย)

- ebox เป็นเวกเตอร์ที่ประกอบไปด้วยพารามิเตอร์หกตัวดังนี้

```
ebox = [xmin, xmax, ymin, ymax, zmin, zmax]
```

ซึ่งจะถูกใช้เพื่อกำหนดขอบเขตของรูปกราฟ เมื่อ

- `xmin` และ `xmax` คือ ค่าต่ำสุดและค่าสูงสุดของเส้นแกน `x`
- `ymin` และ `ymax` คือ ค่าต่ำสุดและค่าสูงสุดของเส้นแกน `y`
- `zmin` และ `zmax` คือ ค่าต่ำสุดและค่าสูงสุดของเส้นแกน `z`

โดยทั่วไปแล้ว `ebox` มักใช้ในกรณีที่ `flag` ถูกกำหนดให้มีค่าเป็น `1`, `3`, และ `5` ถ้าไม่มีการใช้ `flag` ในคำสั่ง `plot3d` พารามิเตอร์ `ebox` จะถูกเพิกเฉย

ตัวอย่างที่ 2 กำหนดให้ตัวแปร `x` และ `y` มีค่าระหว่าง `0` ถึง `1` จงวาดกราฟสามมิติจากสมการ

$$z = \left|0.5\cos\left(2x\pi\right)\ \cos\left(2y\pi\right)\right|$$

วิธีทำ จากโจทย์ สามารถวาดกราฟสามมิติได้โดยใช้ชุดคำสั่งของโปรแกรม SCILAB ดังนี้

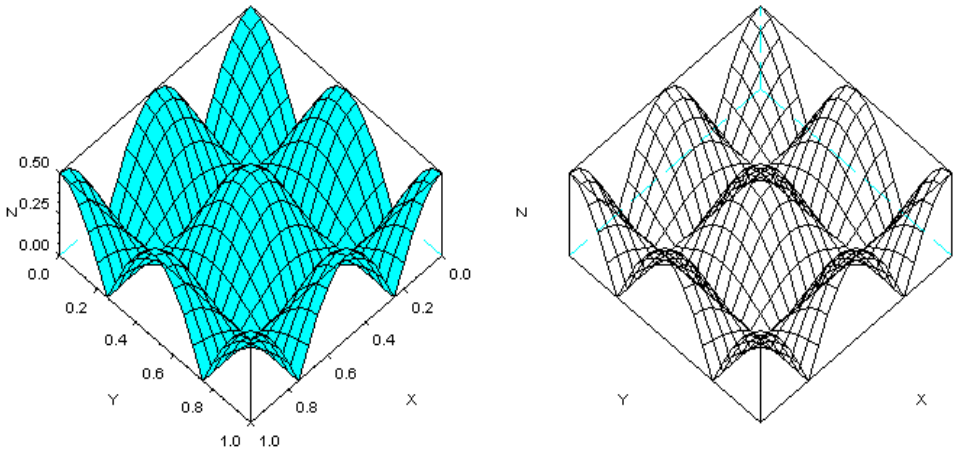
```
-->clf;  x = linspace(0, 1, 21);
-->y = linspace(0, 1, 21);
-->z = abs(0.5 * cos(2*%pi*x)' * cos(2*%pi*y));
-->subplot(1, 2, 1); plot3d(x, y, z, flag = [4, 2, 4]);      //รูปที่ 6.20 ด้านซ้าย
-->subplot(1, 2, 2); plot3d(x, y, z, alpha = 35, theta = 45, ...
-->flag = [0, 2, 3]);                                        //รูปที่ 6.20 ด้านขวา
```

ผลลัพธ์ที่ได้จากชุดคำสั่งเหล่านี้แสดงในรูปที่ 6.20[25]

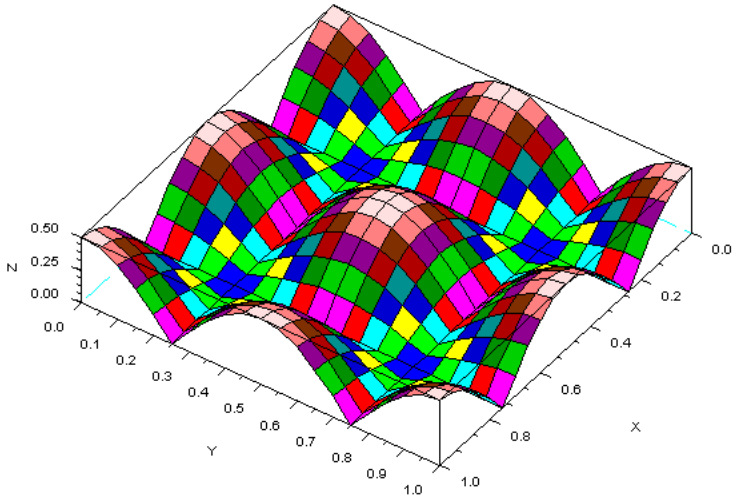
จากรูปที่ 6.20 ด้านซ้าย ถ้าต้องการให้มีการไล่โทนสีตามขนาดของค่าในแกน `z` ก็สามารถทำได้โดยการใช้คำสั่ง `plot3d1` ดังนี้

```
-->clf;  plot3d1(x, y, z, theta = 35, alpha = 25, flag = [0, 2, 4]);
```

[25] รูปที่ 6.20 ด้านขวามีรูปร่างลักษณะเหมือนกับรูปพื้นผิวร่างแห (mesh surface plot) ดังนั้นผู้ใช้สามารถวาดรูปพื้นผิวร่างแหได้จากคำสั่งนี้เช่นกัน



รูปที่ 6.20 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `plot3d`



รูปที่ 6.21 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `plot3d1`

ซึ่งจะได้ผลลัพธ์ตามรูปที่ 6.21

นอกจากนี้ถ้าต้องการทราบความสัมพันธ์ระหว่างสีกับขนาดของค่าในแกน z ก็สามารถทำได้โดยใช้คำสั่ง `colorbar` ซึ่งมีรูปแบบการใช้งานคือ

```
colorbar(umin, umax, [colminmax])
```

โดยที่พารามิเตอร์

- umin เป็นเลขจำนวนจริงของค่าต่ำสุดของขนาดของค่าในแกน z
- umax เป็นเลขจำนวนจริงของค่าสูงสุดของขนาดของค่าในแกน z
- colminmax เป็นตัวเลือกที่มีรูปแบบการใช้งาน คือ

```
colminmax = [1  nb_colors]
```

เป็นเวกเตอร์ขนาด 1x2 โดยที่ nb_colors คือจำนวนสีที่จะใช้ในรูปกราฟ

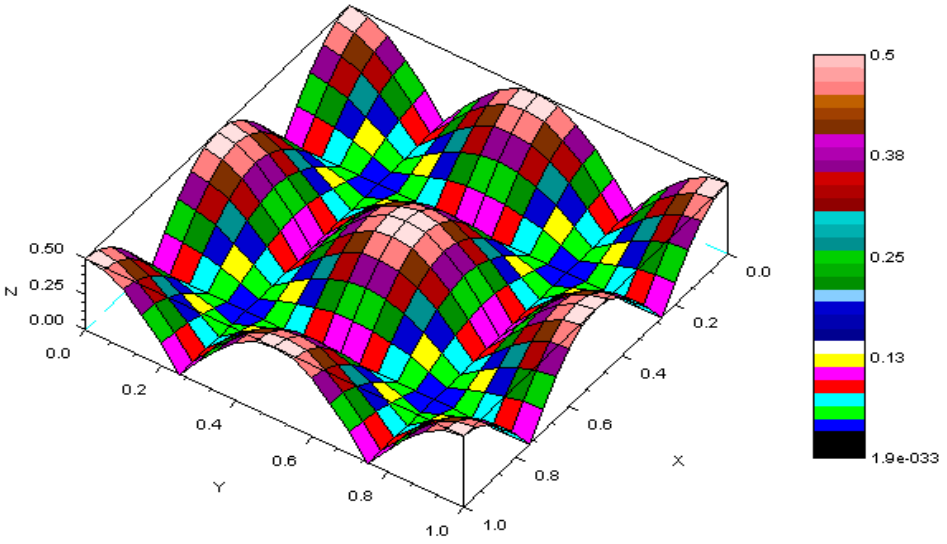
ตัวอย่างการใช้งานคำสั่งนี้ เช่น (ต่อเนื่องจากรูปที่ 6.21)

```
-->clf;  zmin = min(z);
-->zmax = max(z);
-->colorbar(zmin, zmax, [1 30]);
-->plot3d1(x, y, z, theta=35, alpha=25, flag=[0, 2, 4]);
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.22 ซึ่งจะมีแถบสีแสดงความสัมพันธ์ระหว่างสีต่างๆ กับขนาดของค่าในแกน z

นอกจากนี้โปรแกรม SCILAB ยังมีคำสั่งที่ใช้ในการวาดภาพสามมิติแบบอื่นอีกมากมาย ตามที่แสดงในตารางที่ 6.6 ตัวอย่างการใช้งานคำสั่งเหล่านี้เช่น

```
-->clf;  u = linspace(-%pi/2, %pi/2, 20);
-->v = linspace(0, 2*%pi, 10);
-->X = cos(u)'*cos(v);
-->Y = cos(u)'*sin(v);
-->Z = sin(u)'*ones(v);
-->subplot(2, 2, 1);
-->plot3d2(X, Y, Z);  xtitle('plot3d2');          //รูปที่ 6.23 ด้านบนซ้าย
```

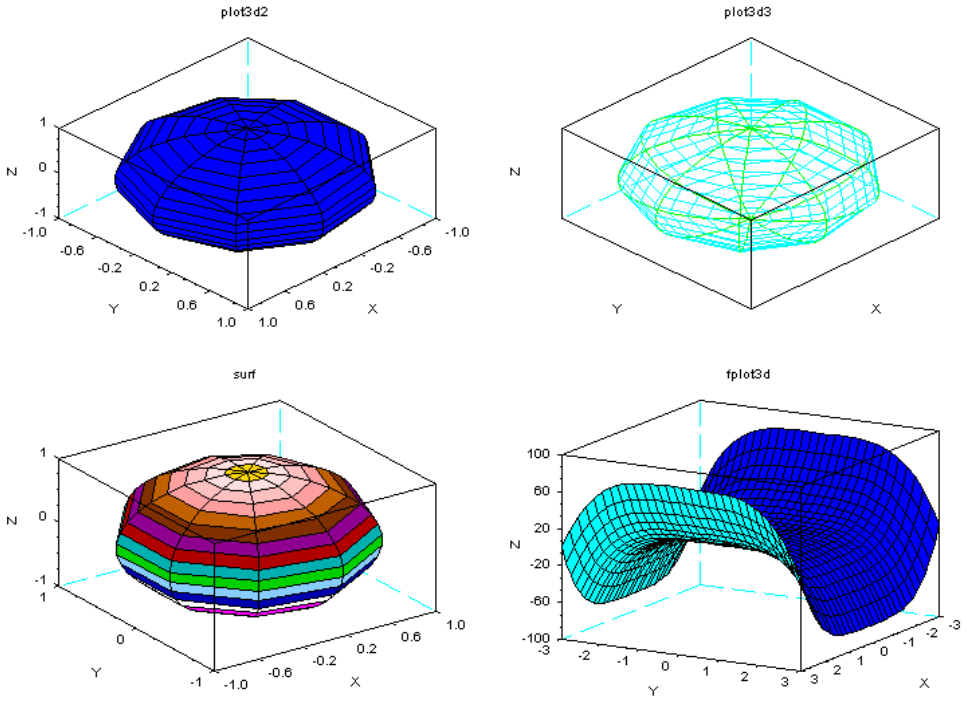


รูปที่ 6.22 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `colorbar` ร่วมกับ `plot3d1`

ตารางที่ 6.6 ตัวอย่างคำสั่งในการวาดกราฟสามมิติแบบทั่วไป

คำสั่ง	คำอธิบาย
`plot3d2`	วาดกราฟพื้นผิวแบบมีสี
`plot3d3`	วาดกราฟพื้นผิวร่างแห (mesh surface plot)
`surf`	วาดกราฟพื้นผิวร่างแห โดยมีการไล่โทนสีตามขนาดของค่าในแกน z
`fplot3d`	วาดกราฟพื้นผิวที่กำหนดโดยฟังก์ชันทางคณิตศาสตร์
`fplot3d1`	วาดกราฟพื้นผิวที่กำหนดโดยฟังก์ชันทางคณิตศาสตร์ โดยมีการไล่โทนสีตามขนาดของค่าในแกน z

```
-->subplot(2, 2, 2);
-->plot3d3(X, Y, Z);  xtitle('plot3d3');          //รูปที่ 6.23 ด้านบนขวา
-->subplot(2, 2, 3);
-->surf(X, Y, Z);  xtitle('surf');                //รูปที่ 6.23 ด้านล่างซ้าย
-->deff('z=f(x, y)', 'z = x^4 - y^4');            //สร้างฟังก์ชันโดยใช้คำสั่ง deff
-->x = -3:0.2:3;
```



รูปที่ 6.23 ตัวอย่างรูปกราฟสามมิติแบบทั่วไป

```
-->y = x;
-->subplot(2, 2, 4);
-->fplot3d(x, y, f, alpha=5, theta=30);          //รูปที่ 6.23 ด้านล่างขวา
-->xtitle('fplot3d');
```

ผลลัพธ์ที่ได้จากชุดคำสั่งเหล่านี้แสดงในรูปที่ 6.23 สำหรับผู้สนใจสามารถศึกษารายละเอียดการใช้งานคำสั่งเหล่านี้เพิ่มเติมได้ในคำสั่ง `help`

6.3.2 กราฟคอนทัวร์

กราฟคอนทัวร์ (contour plot) หรือกราฟเส้นชั้นความสูง เป็นกราฟที่เกิดจากการเฉือนรูปกราฟสามมิติแล้วมองจากด้านบนลงมาจะได้ออกมาเป็นเส้นชั้นความสูง โดยแต่ละเส้นชั้นความสูงที่มีสี

เดียวกันจะแสดงถึงระดับความสูงที่เท่ากันในรูปกราฟสามมิติ การวาดรูปกราฟคอนทัวร์ในโปรแกรม SCILAB สามารถทำได้โดยการใช้คำสั่ง

```
contour(x, y, z, nz, [options])
```

โดยที่พารามิเตอร์ x, y, และ z แสดงค่าในเส้นแกน x, เส้นแกน y, และเส้นแกน z ตามลำดับ ส่วนพารามิเตอร์ nz เป็นเลขจำนวนเต็มบวกที่ใช้กำหนดจำนวนระดับความสูง (หรือจำนวนเส้น) ที่ต้องการให้แสดงในรูปกราฟคอนทัวร์ และพารามิเตอร์ options จะมีรูปแบบการใช้งานคือ

```
options = [theta, alpha, leg, flag, ebox, zlev]
```

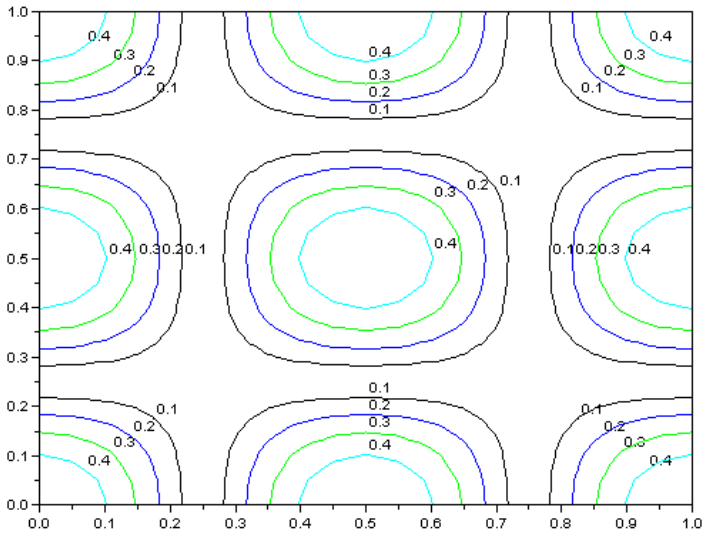
เมื่อ

- theta, alpha, leg, flag, ebox มีรูปแบบเหมือนกับที่กล่าวมาแล้วข้างต้น
- zlev เป็นเลขจำนวนจริงที่ใช้ในการปรับขนาดค่าของเส้นแกน z กล่าวคือค่าของสมาชิกทุกสมาชิกในตัวแปรของเส้นแกน z จะถูกนำไปบวกรวมกับค่า zlev ก่อนที่จะนำค่าของผลลัพธ์ที่ได้ไปใช้กำหนดขนาดค่าของเส้นแกน z ในการแสดงผลของรูปกราฟ

ตัวอย่างเช่น ถ้าต้องการวาดรูปคอนทัวร์ของกราฟสามมิติในรูปที่ 6.21 ก็สามารถทำได้โดยการใช้คำสั่งต่อไปนี้

```
-->clf;  x = linspace(0, 1, 21);
-->y = linspace(0, 1, 21);
-->z = abs(0.5*cos(2*%pi*x)'*cos(2*%pi*y));
-->contour(x, y, z, 4)
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.24 สังเกตจะพบว่าส่วนที่นูนขึ้นมาแต่ละอันในรูปที่ 6.21 จะแสดงด้วยเส้นสีสี่เส้น (เนื่องจาก nz = 4) โดยที่แต่ละเส้นสีจะแสดงค่าระดับความสูง (หรือขนาดของค่าในแกน z) ที่เท่ากัน



รูปที่ 6.24 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้คำสั่ง `plot3d1`

ตารางที่ 6.7 ตัวอย่างคำสั่งวาดกราฟสามมิติแบบพิเศษสำหรับการใช้งานเฉพาะด้าน

คำสั่ง	คำอธิบาย
`param3d`	วาดเส้นโค้งแบบอิงพารามิเตอร์ (parametric curve) หนึ่งเส้น
`param3d1`	วาดเส้นโค้งแบบอิงพารามิเตอร์หลายๆ เส้นในรูปกราฟเดียวกัน
`hist3d`	วาดกราฟฮีสโตแกรมแบบสามมิติ (3D histogram)
`fcontour`	วาดกราฟคอนทัวร์ที่กำหนดโดยฟังก์ชันทางคณิตศาสตร์

6.3.3 การวาดกราฟสามมิติแบบพิเศษ

นอกจากนี้โปรแกรม SCILAB ยังได้เตรียมคำสั่งจำนวนมากสำหรับการวาดกราฟสามมิติแบบพิเศษไว้ใช้ในงานเฉพาะด้านโดยเฉพาะงานทางด้านวิศวกรรมและวิทยาศาสตร์ ดังแสดงในตารางที่ 6.7 (ผู้สนใจสามารถศึกษารายละเอียดเพิ่มเติมของคำสั่งวาดกราฟสามมิติเหล่านี้ได้จากคำสั่ง `help`) ตัวอย่างการใช้งานคำสั่งเหล่านี้ เช่น

```
-->clf;  t = [0:0.1:5*%pi]';
-->subplot(2, 2, 1);
```

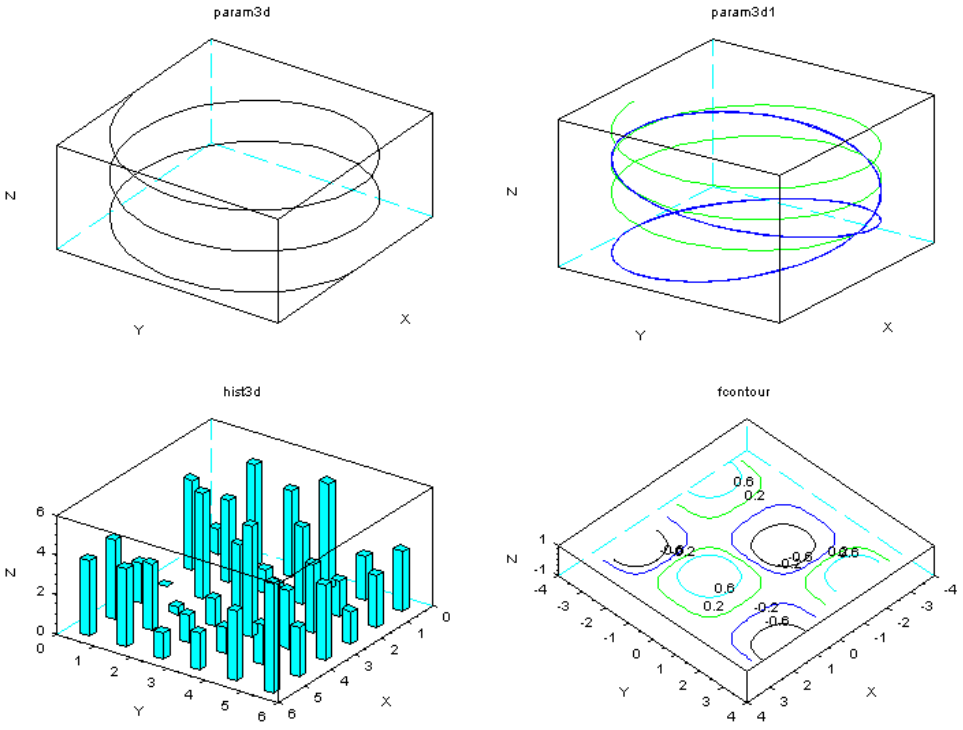
```
-->param3d(sin(t), cos(t), t/10, 35, 45, "X@Y@Z", [2,3]);
-->xtitle('param3d');                                    //รูปที่ 6.25 ด้านบนซ้าย
-->subplot(2, 2, 2);
-->param3d1([sin(t), sin(2*t)],  [cos(t),cos(2*t)], ...
-->list([t/10, sin(t)], [3,2]), 35, 45, "X@Y@Z", [2,3]);
-->xtitle('param3d1');                                   //รูปที่ 6.25 ด้านบนขวา
-->subplot(2, 2, 3);
-->hist3d(6*rand(6,6));
-->xtitle('hist3d');                                     //รูปที่ 6.25 ด้านล่างซ้าย
-->deff("[z] = f(x, y)", "z = sin(x)*cos(y)");
-->t = %pi*[-10:10]/10;
-->subplot(2, 2, 4);
-->fcontour(t, t, f, 4, ebox = [-4  4  -4  4  -1  1], flag = [0 1 4]);
-->xtitle('fcontour');                                   //รูปที่ 6.25 ด้านล่างขวา
```

ผลลัพธ์ที่ได้จากชุดคำสั่งเหล่านี้แสดงในรูปที่ 6.25

6.4 การวาดกราฟแบบผสม

ในการวาดกราฟบางครั้งมีความจำเป็นต้องวาดกราฟสองแบบผสมกัน เพื่อที่จะได้สามารถอธิบายความหมายของรูปกราฟนั้นได้ครบถ้วน โปรแกรม SCILAB สามารถที่จะวาดกราฟลักษณะนี้ได้โดยการวาดกราฟทีละรูปลงไปทับซ้อนกัน ดังแสดงในตัวอย่างต่อไปนี้

```
-->clf;  x = linspace(0, 1, 21);
-->y = linspace(0, 1, 21);
-->z = 2*abs(cos(2*%pi*x)'*cos(2*%pi*y));
-->rect = [min(x), max(x), min(y), max(y), min(z)-5, max(z)+1];
-->plot3d(x, y, z, theta = 35, alpha = 15, flag = [2,1,4], ebox = rect);
```



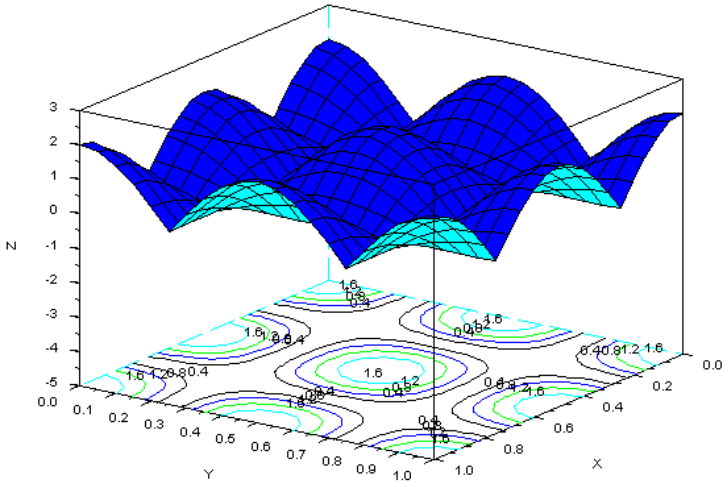
รูปที่ 6.25 ตัวอย่างรูปกราฟแสดงผลลัพธ์จากการใช้ชุดคำสั่งสำหรับวาดกราฟสามมิติที่ใช้งานเฉพาะด้าน

```
-->contour(x, y, z, 4, theta = 35, alpha = 15, flag = [1,1,4], ...
-->ebox = rect, zlev = -5);
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.26 ซึ่งจะเห็นได้ว่าการวาดกราฟแบบผสมนี้ไม่ยุ่งยาก ดังนั้นผู้ใช้สามารถที่จะวาดกราฟแบบผสมใดๆ ก็ได้โดยใช้คำสั่งวาดกราฟต่างๆ ตามที่ได้อธิบายไว้ข้างต้น เพื่อที่จะได้รูปกราฟในแบบที่ต้องการ

6.5 ตัวอย่างการวาดกราฟ

ในส่วนนี้จะยกตัวอย่างการวาดกราฟแบบต่างๆ เพื่อให้เข้าใจถึงขั้นตอนในการวาดกราฟที่มีคุณสมบัติตามที่ระบุไว้ ดังต่อไปนี้



รูปที่ 6.26 ตัวอย่างการวาดรูปกราฟแบบผสม

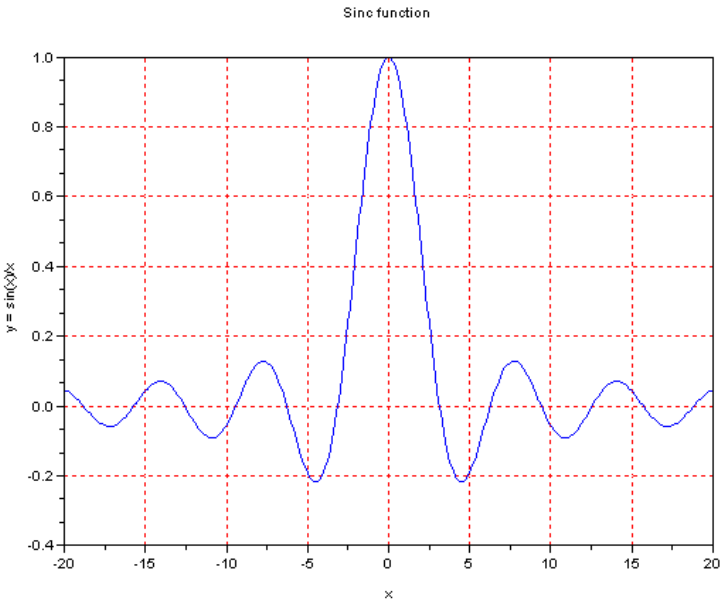
ตัวอย่างที่ 3 ในเรื่องการสื่อสารดิจิทัล (digital communication) ฟังก์ชันของรูปสัญญาณที่พบบ่อย คือฟังก์ชันซิงก์ (sinc function) ซึ่งมีรูปแบบตามสมการ

$$y = \frac{\sin(x)}{x}$$

เมื่อ x เป็นระยะห่างจากจุดกำเนิด (origin point) ถ้ากำหนดให้ $-20 < x < 20$ จงวาดรูปกราฟของฟังก์ชันซิงก์นี้

วิธีทำ กราฟของฟังก์ชันซิงก์สามารถที่จะวาดโดยใช้โปรแกรม SCILAB ได้ ดังนี้

```
-->clf;
-->x = -20:0.1:20;
-->x(find(x == 0)) = %eps;
-->y = sin(x)./x;
-->plot(x,y); xtitle("Sinc function","x","y = sin(x)/x");
-->xgrid(5);                                    //ใส่เส้นกริดสีแดงในหน้าต่างกราฟ
```



รูปที่ 6.27 รูปสัญญาณของฟังก์ชันซิงก์ `y = sin(x)/x`

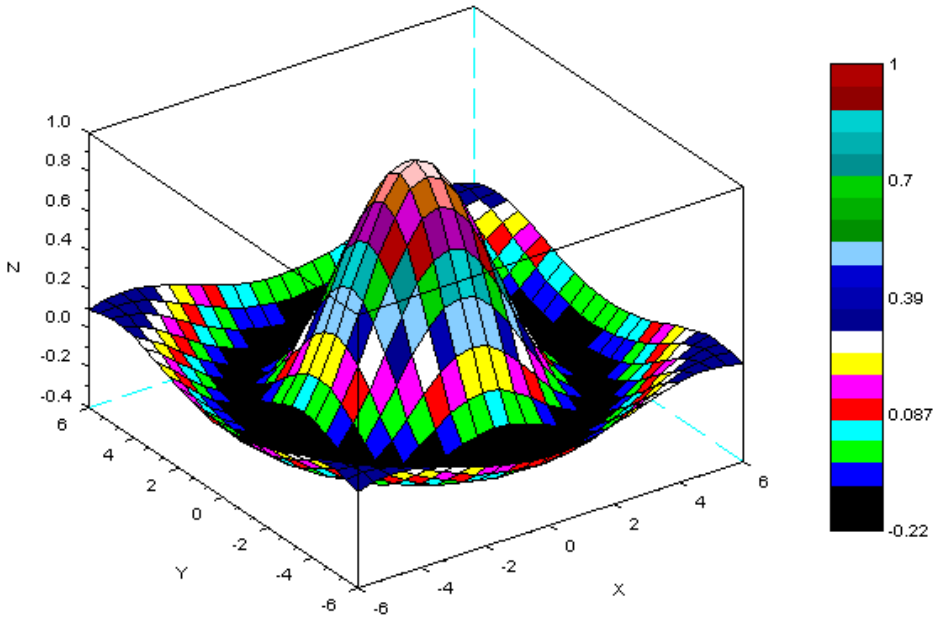
ในชุดคำสั่งบรรทัดที่สาม คำสั่ง `find` จะทำหน้าที่หาตำแหน่งของตัวแปร `x` ที่มีค่าเป็นศูนย์ว่าอยู่ตำแหน่งไหน จากนั้นก็เปลี่ยนค่าของตัวแปร `x` ที่มีค่าเป็นศูนย์ให้มีค่าเท่ากับ `%eps` แทน เพื่อป้องกันการหารด้วยค่า 0 ของฟังก์ชัน `y` ผลลัพธ์ที่ได้จากชุดคำสั่งนี้แสดงในรูปที่ 6.27

ตัวอย่างที่ 4 ฟังก์ชันซิงก์แบบสามมิติสามารถแสดงได้ด้วยสมการ

$$y = \frac{\sin(d)}{d} \quad \text{และ} \quad d = \sqrt{x^2 + y^2}$$

โดยที่ d คือระยะทางจากจุดศูนย์กลางไปยังจุดใดๆ ในพิกัดทรงกลม (ดูรูปที่ 6.19) จงวาดรูปกราฟของฟังก์ชันซิงก์แบบสามมิติ ถ้ากำหนดให้ $-6 < x < 6$ และ $-6 < y < 6$ และต้องการให้รูปกราฟที่ได้มีการไล่โทนสีและมีแถบสีแสดงความสัมพันธ์ระหว่างสีต่างๆ กับขนาดของค่าในเส้นแกน z

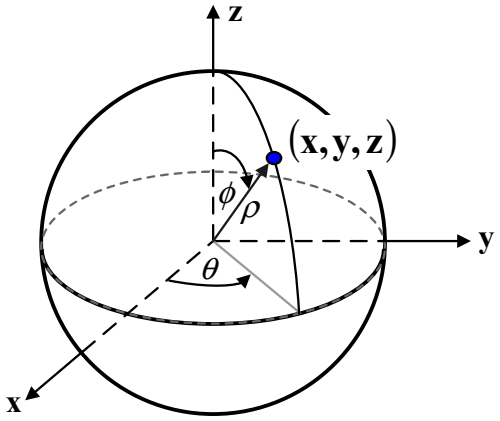
วิธีทำ รูปกราฟของฟังก์ชันซิงก์แบบสามมิติสามารถที่จะวาดโดยใช้โปรแกรม SCILAB ได้ดังนี้



รูปที่ 6.28 รูปสัญญาณของฟังก์ชันซิงก์แบบสามมิติ

```
-->clf;  x = -6:0.5:6;
-->y = -6:0.5:6;
-->[XX, YY] = ndgrid(x, y);
-->d = sqrt(XX.^2 + YY.^2);
-->d(find(d == 0)) = %eps;
-->ZZ = sin(d) ./ d;
-->zmin = min(ZZ);
-->zmax = max(ZZ);
-->colorbar(zmin, zmax, [1, 20]);
-->surf(XX, YY, ZZ);
```

คำสั่ง `ndgrid` (บรรทัดที่สามของชุดคำสั่งนี้) จะทำหน้าที่ในการสร้างจุดพิกัด `(x,y)` ทั้งหมดในช่วงค่าของเส้นแกน `x` และเส้นแกน `y` ผลลัพธ์ที่ได้จากชุดคำสั่งนี้แสดงในรูปที่ 6.28



รูปที่ 6.29 รูปแสดงพิกัดทรงกลม

ตัวอย่างที่ 5 จงวาดรูปกราฟพื้นผิวแบบมีสีของรูปทรงกลม (sphere) บนระบบพิกัดคาร์ทีเซียน x–y–z ที่มีจุดศูนย์กลางของทรงกลมที่จุดพิกัด (x, y, z) = (0, 0, 0) และมีความยาวรัศมี (radius) เท่ากับ 1 หน่วย

วิธีทำ ก่อนอื่นให้พิจารณาสมการของทรงกลมบนระบบพิกัดคาร์ทีเซียนดังนี้

$$x^2 + y^2 + z^2 = \rho^2$$

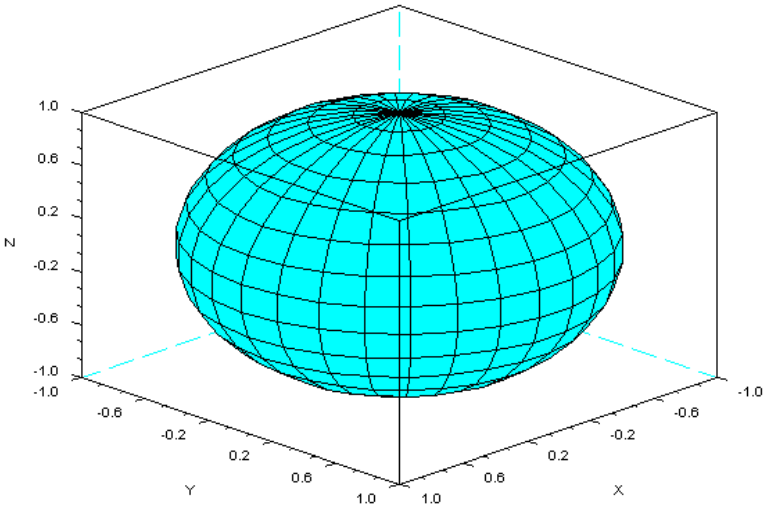
โดยที่ (x, y, z) คือจุดพิกัดในระบบพิกัดคาร์ทีเซียน หากต้องการแปลงสมการของทรงกลมบนระบบพิกัดคาร์ทีเซียนให้อยู่ในระบบพิกัดทรงกลม ก็สามารถทำได้ดังนี้ จากรูปที่ 6.29 จะได้ว่า

$$x = \rho\cos(\theta)\sin(\phi)$$

$$y = \rho\sin(\theta)\sin(\phi)$$

$$z = \rho\cos(\phi)$$

โดยที่ ρ คือความยาวรัศมีของทรงกลม, θ คือมุมกวาด (azimuth) หรือมุมที่เทียบกับแกน x มีค่าระหว่าง 0 ถึง 2π, และ ϕ คือมุมยก (elevation) หรือมุมที่เทียบกับแกน z มีค่าระหว่าง 0 ถึง π



รูปที่ 6.30 ตัวอย่างรูปกราฟทรงกลม

ดังนั้นเมื่อเข้าใจถึงความสัมพันธ์ระหว่างระบบพิกัดคาร์ทีเซียนและระบบพิกัดทรงกลมแล้ว ก็จะทำให้สามารถวาดรูปทรงกลมที่มีความยาวรัศมีเท่ากับ 1 หน่วยได้โดยใช้ชุดคำสั่งของโปรแกรม SCILAB ดังนี้

```
-->clf;
-->phi = linspace(0, %pi, 16);
-->theta = linspace(0, 2*%pi, 31);
-->rho = 1;
-->[PHI, THETA] = ndgrid(phi, theta);
-->X = rho * cos(THETA) .* sin(PHI);
-->Y = rho * sin(THETA) .* sin(PHI);
-->Z = rho * cos(PHI);
-->plot3d2(X,Y,Z,alpha=60)
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.30

ตัวอย่างที่ 6 จงวาดรูปกราฟพื้นผิวแบบมีสีของรูปทรงห่วงยาง (torus) ที่มีลักษณะเหมือนกับขนมโดนัตบนระบบพิกัดคาร์ทีเซียน โดยกำหนดให้มีความยาวรัศมีของรูปทรงห่วงยางในระนาบ x–y เท่ากับ 1.4 หน่วย และในแนวแกน z เท่ากับ 0.5 หน่วย และจุดศูนย์กลางของรูปทรงห่วงยางอยู่ที่จุดพิกัด (x, y, z) = (0, 0, 0)

วิธีทำ ก่อนอื่นให้พิจารณาสมการของรูปทรงห่วงยางบนระบบพิกัดคาร์ทีเซียน ดังนี้

$$\left(c-\sqrt{x^2+y^2}\right)^2+z^2=a^2$$

เมื่อ (x, y, z) คือจุดพิกัดในระบบพิกัดคาร์ทีเซียน, a คือความยาวรัศมีของรูปทรงห่วงยางในระนาบ x–y, และ c คือความยาวรัศมีของวงกลมในแนวแกน z เนื่องจากการเตรียมข้อมูลเพื่อวาดรูปทรงห่วงยางในระบบพิกัดอื่นจะง่ายกว่าการเตรียมข้อมูลเพื่อวาดรูปในระบบพิกัดคาร์ทีเซียน ดังนั้นข้อมูลของตัวแปร x, y, และ z ควรที่จะต้องถูกแปลงให้อยู่ในรูปของตัวแปรใหม่ที่เหมาะสมดังนี้

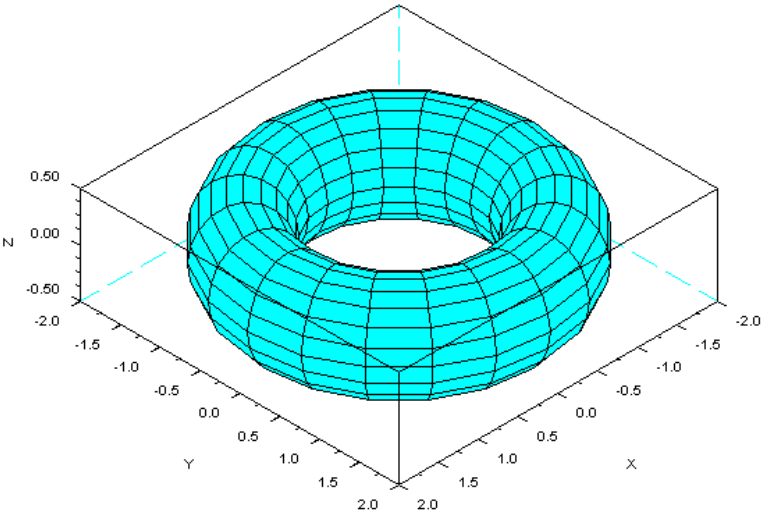
$$x=\left(a+c\cos\left(\phi\right)\right)\cos\left(\theta\right)$$

$$y=\left(a+c\cos\left(\phi\right)\right)\sin\left(\theta\right)$$

$$z=c\sin\left(\phi\right)$$

โดยที่ตัวแปร θ และ ϕ จะมีค่าอยู่ระหว่าง 0 ถึง 2π การแปลงสมการแบบนี้จะเรียกว่า สมการอิงพารามิเตอร์ (parametric equation) จากนั้นก็สามารถวาดรูปทรงห่วงยางตามที่โจทย์ต้องการได้ โดยใช้ชุดคำสั่งของโปรแกรม SCILAB ดังนี้

```
-->clf;  phi = linspace(0, 2*%pi, 21);
-->theta = linspace(0, 2*%pi, 21);
-->a = 1.4;
-->c = 0.5;
-->[PHI, THETA] = ndgrid(phi, theta);
-->X = (a + c*cos(PHI)) .* cos(THETA);
```



รูปที่ 6.31 ตัวอย่างรูปกราฟทรงห่วงยาง

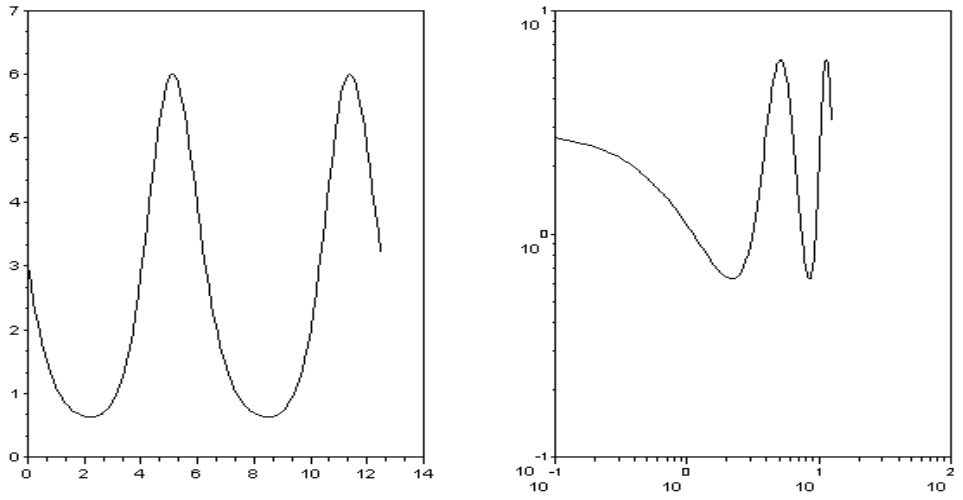
```
-->Y = (a + c * cos(PHI)) .* sin(THETA);
-->Z = c * sin(PHI);
-->plot3d2(X, Y, Z, alpha = 60, flag = [4, 2, 4]);
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.31

ตัวอย่างที่ 7 กำหนดให้ $f(x) = (2+\cos(x))e^{-\sin(x)}$ เมื่อ $0 \le x \le 4\pi$ จงวาดกราฟเชิงเส้นและกราฟลอการิทึม (ทั้งในแนวแกน x และแกน y)

วิธีทำ กราฟเชิงเส้นและกราฟลอการิทึมสามารถหาได้จากชุดคำสั่งดังนี้

```
-->clf;  x = 0:0.1:4*%pi;
-->y = (2 + cos(x)).*exp(-sin(x));
-->subplot(1, 2, 1);   plot2d(x, y);                    //รูปที่ 6.32 ด้านซ้าย
-->subplot(1, 2, 2);   plot2d(x, y, logflag="ll");      //รูปที่ 6.32 ด้านขวา
```



รูปที่ 6.32 รูปกราฟของฟังก์ชัน $f(x) = (2+\cos(x))e^{-\sin(x)}$

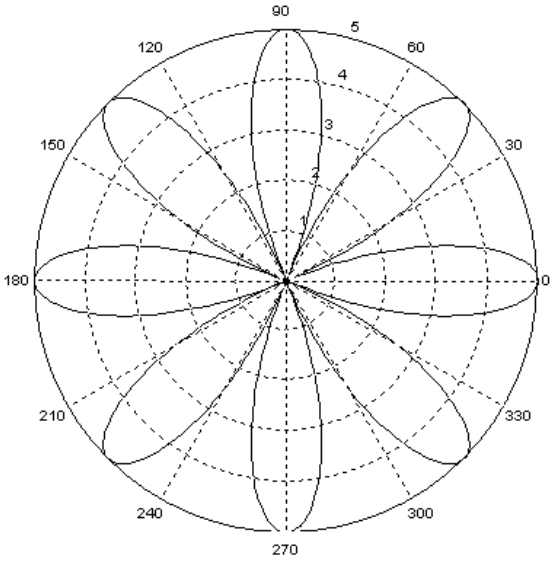
ผลลัพธ์ที่ได้แสดงในรูปที่ 6.32

ตัวอย่างที่ 8 วาดกราฟเส้นโค้งกลีบกุหลาบ (rose curve) ที่กำหนดโดยสมการ $r = 5\cos(4\theta)$ เมื่อ $0 \le \theta \le 2\pi$ ในระบบพิกัดเชิงขั้ว

วิธีทำ กราฟเส้นโค้งกลีบกุหลาบนี้สามารถหาได้จากชุดคำสั่งดังนี้

```
-->clf;
-->theta = 0:0.01:2*%pi;
-->r = 5*cos(4*theta);
-->polarplot(theta, r)
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 6.33



รูปที่ 6.32 รูปกราฟเส้นโค้งกลีบกุหลาบ $r = 5\cos(4\theta)$

6.6 สรุป

ในบทนี้ได้อธิบายถึงการใช้คำสั่งต่างๆ ที่ใช้ในการวาดกราฟสองมิติ (เช่น คำสั่ง `plot`, `plot2d`, `polarplot`, `contour2d`, และ `plzr` เป็นต้น) และกราฟสามมิติ (เช่น คำสั่ง `plot3d`, `countour`, และ `hist3d`) พร้อมทั้งกล่าวถึงพารามิเตอร์ต่างๆ ที่ใช้ในคำสั่งเหล่านี้ เพื่อให้สามารถนำไปประยุกต์ใช้งานได้อย่างถูกต้องและตรงตามความต้องการ

ดังนั้นจะเห็นได้ว่าโปรแกรม SCILAB สามารถวาดกราฟได้หลายรูปแบบจึงเหมาะอย่างยิ่งสำหรับการนำมาใช้ในการวาดรูปกราฟ เพื่อใช้ในการนำเสนอผลงานวิจัยและผลงานทางวิชาการโดยเฉพาะอย่างยิ่งงานทางด้านวิศวกรรมและวิทยาศาสตร์

6.7 แบบฝึกหัดท้ายบท

6.1 กำหนดให้ $y = 4x^2$ โดยที่ $1 \le x \le 100$ จงวาดกราฟที่มีรูปแบบต่างๆ ดังนี้

6.1.1) กราฟเชิงเส้น

6.1.2) กราฟลอการิทึมในแนวแกน x

6.1.3) กราฟลอการิทึมในแนวแกน y

6.1.4) กราฟลอการิทึมทั้งในแนวแกน x และแกน y

6.2 กำหนดให้ $y = 10 \log_{10}(3x)$ โดยที่ $1 \le x \le 100$

6.2.1) วาดกราฟเชิงเส้น

6.2.2) วาดกราฟลอการิทึมทั้งในแนวแกน x และแกน y

6.2.3) เปรียบเทียบข้อแตกต่างของกราฟที่ได้ทั้งสองรูป

6.3 กำหนดให้ $y = (e^x - e^{-x})/2$ โดยที่ $-10 \le x \le 10$

6.3.1) วาดกราฟเชิงเส้น

6.3.2) วาดกราฟลอการิทึมทั้งในแนวแกน x และแกน y

6.3.3) เปรียบเทียบข้อแตกต่างของกราฟที่ได้ทั้งสองรูป

6.4 จงวาดกราฟของภาพตัดกรวยจากสมการต่อไปนี้

6.4.1) $x^2 + y^2 - 9 = 0$

6.4.2) $x^2 + y^2 - 8x + 10y + 5 = 0$

6.4.3) $x^2 - 4x - 2y + 10 = 0$

6.4.4) $y^2 - 2y - 8x + 9 = 0$

6.4.5) $y^2 + 4y + 20x + 4 = 0$

6.4.6) $x^2 + 4y^2 - 64 = 0$

6.4.7) $9x^2 + y^2 - 18 = 0$

6.4.8) $4x^2 + y^2 + 8x - 4y - 8 = 0$

6.4.9) $5x^2 - 2y^2 - 10 = 0$

6.4.10) $9x^2 - 16y^2 - 90x + 64y + 17 = 0$

6.4.11) $7y^2 - 9x^2 - 70y - 54x + 31 = 0$

6.4.12) $17x^2 + 12xy + 8y^2 - 20 = 0$

6.4.13) $3x^2 - xy + 4y^2 - 16x - 3y = 0$

6.4.14) $153x^2 - 192xy + 97y^2 - 30x - 40y - 200 = 0$

6.5 เส้นโค้งลิมาคอน (Limacons) ถูกกำหนดโดยสมการ $r = a \pm b\sin(\theta)$ และ $r = a \pm b\cos(\theta)$ เมื่อ a และ b เป็นเลขจำนวนจริง และ $0 \le \theta \le 2\pi$ จงวาดกราฟของสมการต่อไปนี้ในระบบพิกัดเชิงขั้ว

6.5.1) $r = 2 + 2\sin(\theta)$

6.5.2) $r = 4 + 2\sin(\theta)$

6.5.3) $r = 2 + 4\sin(\theta)$

6.5.4) จงอธิบายรูปกราฟที่ได้จากข้อ 6.5.1, 6.5.2, และ 6.5.3

6.5.5) $r = 2 - 2\cos(\theta)$

6.5.6) $r = 4 - 2\cos(\theta)$

6.5.7) $r = 2 - 4\cos(\theta)$

6.5.8) จงอธิบายรูปกราฟที่ได้จากข้อ 6.5.5, 6.5.6, และ 6.5.7

6.6 เส้นโค้งกลีบกุหลาบ (rose curve) ถูกกำหนดโดยสมการ $r = a\sin(n\theta)$ และ $r = a\cos(n\theta)$ โดยที่ a เป็นเลขจำนวนจริง, n เป็นเลขจำนวนเต็มบวก, และ $0 \le \theta \le 2\pi$ จงวาดกราฟของสมการต่อไปนี้ในระบบพิกัดเชิงขั้ว

6.6.1) $r = 3\sin(2\theta)$

6.6.2) $r = 3\sin(3\theta)$

6.6.3) $r = 3\sin(4\theta)$

6.6.4) $r = 3\cos(3\theta)$

6.6.5) $r = 3\cos(4\theta)$

6.6.6) จงอธิบายรูปกราฟที่ได้จากข้อ 6.6.1, 6.6.2, 6.6.3, 6.6.4, และ 6.6.5

6.7 เส้นโค้งเลมนิสเคต (Lemniscates) ถูกกำหนดโดยสมการ $r^2 = a^2\sin(2\theta)$ และ $r^2 = a^2\cos(2\theta)$ เมื่อ a เป็นเลขจำนวนจริง และ $0 \le \theta \le 2\pi$ จงวาดกราฟของสมการต่อไปนี้ในระบบพิกัดเชิงขั้ว

6.7.1) $r^2 = 4\sin(2\theta)$

6.7.2) $r^2 = 9\sin(2\theta)$

6.7.3) $r^2 = 4\cos(2\theta)$

6.7.4) $r^2 = 9\cos(2\theta)$

6.8 เส้นเวียนก้นหอย (spirals) ถูกกำหนดโดยสมการ $r = a\theta$, $r = e^{a\theta}$, และ $r\theta = a$ โดยที่ a เป็นเลขจำนวนจริง และ $0 \le \theta \le 2\pi$ จงวาดกราฟของสมการต่อไปนี้ในระบบพิกัดเชิงขั้ว

6.8.1) $r = 2\theta$

6.8.2) $\log_e(r) = 2\theta$

6.8.3) $r\theta = 2$

6.9 จงแปลงสมการต่อไปนี้ให้เป็นสมการระบบเชิงขั้ว พร้อมทั้งวาดกราฟเชิงขั้ว

6.9.1) $x^2 - 6x + y^2 + 4y = 0$

6.9.2) $x^2 + y^2 - 2x + 2y = 0$

6.9.3) $y^2 + 4x - 4 = 0$

6.9.4) $x^2 + 2y^2 - 8 = 0$

6.9.5) $\sqrt{x^2 + y^2}\left(\sqrt{x^2 + y^2} - 2\right) - 2y = 0$

6.9.6) $\left(x^2 + y^2\right)^2 - 6x^2y + 2y^3 = 0$

6.10 จงวาดพื้นระนาบจากสมการต่อไปนี้

6.10.1) $x + 2y + 3z = 4$

6.10.2) $2x + 3y - 4z = 5$

6.10.3) $3x - 4y + 5z = 6$

6.10.4) $-2x + 3y - 2z = 1$

6.11 กำหนดให้ $0 \le x \le 1$ และ $0 \le y \le 1$ จงวาดกราฟสามมิติของฟังก์ชันต่อไปนี้

6.11.1) $z = \sin(2\pi x)\cos(3\pi y)$

6.11.2) $z = \cos(2\pi x)\sin(2\pi y)$

6.12 ทำซ้ำข้อ 6.11 แต่วาดกราฟแบบคอนทัวร์ (contour plot)

6.13 ทำซ้ำข้อ 6.11 แต่วาดกราฟแบบพื้นผิวร่างแห (mesh surface plot)

6.14 ทำซ้ำข้อ 6.11 แต่วาดกราฟแบบผสม โดยผสมระหว่างกราฟพื้นผิวและกราฟคอนทัวร์

บทที่ 7

พื้นฐานระบบอินพุตและเอาต์พุต

ระบบอินพุต (input) และเอาต์พุต (output) หรือระบบ I/O เป็นระบบที่เกี่ยวข้องกับการอ่านและเขียนข้อมูลระหว่างโปรแกรม SCILAB กับเครื่องคอมพิวเตอร์ การอ่านข้อมูลหมายถึงการอ่านข้อมูลจากคีย์บอร์ดหรือจากไฟล์ ส่วนการเขียนข้อมูลหมายถึงการพิมพ์ข้อมูลเพื่อแสดงออกทางจอภาพของคอมพิวเตอร์หรือเป็นการเขียนข้อมูลลงไปเก็บไว้ในไฟล์ก็ได้ ในบทนี้จะอธิบายถึงคำสั่งและหลักการในการเขียนโปรแกรม เพื่อใช้ในการอ่านและเขียนข้อมูลระหว่างโปรแกรม SCILAB กับเครื่องคอมพิวเตอร์ ซึ่งจะช่วยทำให้สามารถพัฒนาฟังก์ชันแบบใหม่ๆ ได้

7.1 คำสั่งพื้นฐานสำหรับการติดต่อระบบอินพุตและเอาต์พุต

การเขียนโปรแกรมเพื่อทำหน้าที่ในการอ่านและเขียนข้อมูลระหว่างโปรแกรม SCILAB กับเครื่องคอมพิวเตอร์นั้น สิ่งที่ควรคำนึงถึงคือจะต้องทราบว่ามีคำสั่งอะไรบ้างที่ทำหน้าที่เกี่ยวข้องกับระบบอินพุตและเอาต์พุต รวมทั้งรูปแบบการเรียกใช้งานของคำสั่งเหล่านั้น ฉะนั้นในส่วนนี้จะขออธิบายถึงคำสั่งพื้นฐานในการติดต่อระบบอินพุตและเอาต์พุตดังต่อไปนี้

7.1.1 คำสั่ง `input`

เป็นคำสั่งที่ใช้ในการอ่านค่าจากคีย์บอร์ดเข้ามาเก็บไว้ในตัวแปร เพื่อนำไปประมวลผลในโปรแกรม SCILAB รูปแบบการใช้งานคำสั่งนี้ คือ

```
[x] = input(message, ["string"])
```

โดยที่พารามิเตอร์

- `message` คือ สายอักขระ (character string) ที่ต้องการให้แสดงผลออกทางหน้าต่างคำสั่ง
- `"string"` (หรือ `"s"`) เป็นตัวกำหนดว่า ข้อมูลที่พารามิเตอร์ `x` จะเก็บไว้เป็นสายอักขระ
- `x` คือ ผลลัพธ์ที่ได้จากคีย์บอร์ด ซึ่งเป็นได้ทั้งเลขจำนวนจริงหรือสายอักขระ

ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->x = input("How many iterations?")
How many iterations?-->5
 x  =
    5.
-->y = input("What is your name?", "s")
What is your name?-->Piya Kovintavewat
 y  =
 Piya Kovintavewat
```

7.1.2 คำสั่ง `file`

เป็นคำสั่งที่ใช้ในการเปิดหรือปิดแฟ้มข้อมูลหรือไฟล์ โดยมีรูปแบบการใช้งานทั่วไปดังนี้

```
unit = file('open', filename, [status])
```

โดยที่พารามิเตอร์

- `'open'` เป็นการบอกโปรแกรมให้เปิดไฟล์ `filename` ขึ้นมาใช้งาน
- `filename` เป็นชื่อไฟล์ข้อมูลที่ต้องการจะเรียกขึ้นมาใช้งาน
- `status` เป็นการกำหนดสถานะของไฟล์ที่เปิดขึ้นมาใช้งาน ซึ่งมีอยู่ 4 รูปแบบคือ
  - `"new"` หมายถึงไฟล์ที่เปิดขึ้นมาจะต้องไม่เคยมีอยู่ในสารบบทำงาน
  - `"old"` หมายถึงไฟล์ที่เปิดขึ้นมาจะต้องมีอยู่แล้วในสารบบทำงาน
  - `"unknown"` หมายถึงไฟล์ที่เปิดขึ้นมาจะมีอยู่หรือไม่มีอยู่ในสารบบทำงานก็ได้
  - `"scratch"` หมายถึงไฟล์ที่เปิดขึ้นมาจะถูกลบทิ้งหลังจากเสร็จสิ้นการทำงาน
- `unit` เป็นเลขจำนวนเต็มที่โปรแกรมใช้อ้างถึงชื่อไฟล์ `filename` นั้น

หลังจากเสร็จสิ้นการใช้งานไฟล์ที่เปิดขึ้นมาแล้ว จะต้องทำการปิดการทำงานของไฟล์นั้นด้วยเสมอ โดยใช้คำสั่งดังนี้

```
file('close', unit)
```

ในทางปฏิบัติคำสั่งทั้งสองนี้มักจะใช้งานคู่กันเสมอ นอกจากนี้คำสั่ง `file` ยังสามารถใช้งานในลักษณะอื่นได้อีก (ลองศึกษารายละเอียดการใช้งานคำสั่ง `file` ในคำสั่ง `help`)

7.1.3 คำสั่ง `printf`

เป็นคำสั่งที่ใช้ในการพิมพ์ค่าของตัวแปรออกมาที่หน้าต่างคำสั่ง ซึ่งมีรูปแบบการใช้งานดังนี้

```
printf(format, value_1 , ..., value_n)
```

โดยที่พารามิเตอร์

- `format` คือ สายอักขระที่ต้องการให้แสดงผลออกทางหน้าต่างคำสั่ง
- `value_i` เป็นตัวกำหนดว่าจะให้ข้อมูลใดแสดงผลออกมาที่หน้าต่างคำสั่ง

ตัวอย่างการใช้งานคำสั่งนี้ เช่น

```
-->printf('Result is: alpha = %d", 2.535)        //พิมพ์เลขจำนวนเต็ม
Result is: alpha = 2

-->printf('Result is:\nalpha = %f", 2.535)       //พิมพ์เลขจำนวนจริง
Result is:
alpha = 2.535000
```

สังเกตจะพบว่าในคำสั่งแรกจะมีการใช้รหัสรูปแบบ (format) **`%d`** ซึ่งเป็นตัวบอกว่าให้แสดงข้อมูลออกมาในรูปของจำนวนเต็ม ส่วนรหัสรูปแบบ **`%f`** จะเป็นตัวบอกว่าให้แสดงข้อมูลออกมาในรูปของจำนวนจริง รหัสรูปแบบนี้จะเหมือนกับรหัสรูปแบบทั่วไปที่ใช้ในโปรแกรมภาษาซีตามที่แสดงในตารางที่ 7.1 ส่วนในคำสั่งที่สองจะมีการใช้รหัสบังคับการพิมพ์ (escape sequence) **`\n`** ซึ่งเป็นตัวบอกว่าให้ขึ้นบรรทัดใหม่ รหัสบังคับการพิมพ์นี้จะเหมือนกับรหัสบังคับการพิมพ์ทั่วไปที่ใช้ในโปรแกรมภาษาซีเช่นกันดังแสดงในตารางที่ 7.2

ตารางที่ 7.1 รหัสรูปแบบในโปรแกรม SCILAB

รหัสรูปแบบ	คำอธิบาย
%d	แสดงผลเป็นเลขจำนวนเต็มฐานสิบแบบมีเครื่องหมาย (signed integer)
%u	แสดงผลเป็นเลขจำนวนเต็มฐานสิบแบบไม่มีเครื่องหมาย (unsigned integer)
%x หรือ %X	แสดงผลเป็นเลขจำนวนเต็มฐานสิบหกแบบไม่มีเครื่องหมาย
%f	แสดงผลเป็นเลขจำนวนจริง
%e	แสดงผลเป็นเลขจำนวนจริงในรูปของเลขยกกำลัง
%c	แสดงผลตัวอักขระ
%s	แสดงผลสายอักขระ

ตารางที่ 7.2 รหัสบังคับการพิมพ์ในโปรแกรม SCILAB

รหัสบังคับการพิมพ์	คำอธิบาย
\n	ขึ้นบรรทัดใหม่
\t	แท็ป (tab) ในแนวนอน
\v	แท็ปในแนวตั้ง
\b	เลื่อนเคอร์เซอร์ไปลบตัวอักขระทางซ้ายมือหนึ่งตัวอักขระ
\r	เครื่องหมาย return เหมือนกับการกดปุ่ม Enter
\f	ขึ้นหน้าใหม่
\a	ส่งเสียงดังออกลำโพงหนึ่งครั้ง
\\	เครื่องหมาย \ (backslash)
\'	เครื่องหมาย ' (single quote)
\"	เครื่องหมาย " (double quote)
\?	เครื่องหมาย ? (question mark)
\ooo	พิมพ์ตัวอักขระที่มีเลขฐานแปดตรงกับค่า ooo
\xhh	พิมพ์ตัวอักขระที่มีเลขฐานสอบหกตรงกับค่า hh

ลองศึกษาการใช้งานคำสั่ง printf จากฟังก์ชันที่ใช้สร้างตารางรหัสแอสกี (ASCII code) ในตัวอย่างต่อไปนี้

```
function [] = MyASCIITable()
printf("Generating an ASCII table:\n");
printf("================================================\n");
for j = 19:9:255
    printf('%3d %s %3d %s %3d %s %3d %s %3d %s %3d %s ' + ...
           '%3d %s %3d %s %3d %s %3d %s\n',...
    j,ascii(j),j+1,ascii(j+1),j+2,ascii(j+2),j+3,ascii(j+3),...
    j+4,ascii(j+4),j+5,ascii(j+5),j+6,ascii(j+6),...
    j+7,ascii(j+7),j+8,ascii(j+8),j+9,ascii(j+9))
end
printf("================================================\n");
endfunction
```

หลังจากเขียนฟังก์ชันเสร็จแล้วก็ให้บันทึกลงในไฟล์ที่ชื่อว่า MyASCIITable.sci จากนั้นให้ทำการโหลดชื่อไฟล์นี้เพื่อเรียกใช้งานฟังก์ชัน MyASCIITable โดยใช้คำสั่ง exec หรือ getf ตัวอย่างการใช้งาน เช่น

```
-->getf('MyASCIITable.sci')
-->MyASCIITable
Generating an ASCII table:
============================================================
 19 ‼  20 ¶  21 ┴  22 ┬  23 ┤  24 ↑  25 ├  26 →  27 ←  28
 28    29    30 -  31    32    33    34 "  35 #  36 $  37 %
 37 %  38 &  39 '  40 (  41 )  42 *  43 +  44 ,  45 -  46 .
 46 .  47 /  48 0  49 1  50 2  51 3  52 4  53 5  54 6  55 7
 55 7  56 8  57 9  58 :  59 ;  60 <  61 =  62 >  63 ?  64 @
 64 @  65 A  66 B  67 C  68 D  69 E  70 F  71 G  72 H  73 I
 73 I  74 J  75 K  76 L  77 M  78 N  79 O  80 P  81 Q  82 R
 82 R  83 S  84 T  85 U  86 V  87 W  88 X  89 Y  90 Z  91 [
 91 [  92 \  93 ]  94 ^  95 _  96 `  97 a  98 b  99 c 100 d
100 d 101 e 102 f 103 g 104 h 105 i 106 j 107 k 108 l 109 m
109 m 110 n 111 o 112 p 113 q 114 r 115 s 116 t 117 u 118 v
118 v 119 w 120 x 121 y 122 z 123 { 124 | 125 } 126 ~ 127
127   128 € 129   130 ‚ 131 ƒ 132 „ 133 … 134 † 135 ‡ 136 ^
136 ^ 137 ‰ 138 Š 139 ‹ 140 Œ 141 • 142 Ž 143 • 144 • 145 ‘
145 ‘ 146 ’ 147 “ 148 ” 149 • 150 – 151 — 152 ˜ 153 ™ 154 š
```

```
154 š 155 › 156 œ 157 • 158 ž 159 Ÿ 160    161 ¡ 162 ¢ 163 £
163 £ 164 ¤ 165 ¥ 166 ¦ 167 § 168 ¨ 169 © 170 ª 171 « 172 ¬
172 ¬ 173 - 174 ® 175 ¯ 176 ° 177 ± 178 ² 179 ³ 180 ´ 181 µ
181 µ 182 ¶ 183 · 184 ¸ 185 ¹ 186 º 187 » 188 ¼ 189 ½ 190 ¾
190 ¾ 191 ¿ 192 À 193 Á 194 Â 195 Ã 196 Ä 197 Å 198 Æ 199 Ç
199 Ç 200 È 201 É 202 Ê 203 Ë 204 Ì 205 Í 206 Î 207 Ï 208 Ð
208 Ð 209 Ñ 210 Ò 211 Ó 212 Ô 213 Õ 214 Ö 215 × 216 Ø 217 Ù
217 Ù 218 Ú 219 Û 220 Ü 221 Ý 222 Þ 223 ß 224 à 225 á 226 â
226 â 227 ã 228 ä 229 å 230 æ 231 ç 232 è 233 é 234 ê 235 ë
235 ë 236 ì 237 í 238 î 239 ï 240 ð 241 ñ 242 ò 243 ó 244 ô
244 ô 245 õ 246 ö 247 ÷ 248 ø 249 ù 250 ú 251 û 252 ü 253 ý
253 ý 254 þ 255 ÿ 256    257    258 ┐ 259 └ 260 ┘ 261 │ 262 ─
============================================================
```

โดยที่คำสั่ง `ascii` จะทำหน้าที่แปลงรหัสแอสกีไปเป็นตัวอักขระของโปรแกรม SCILAB หรือให้ในการแปลงตัวอักขระของโปรแกรม SCILAB ไปเป็นรหัสแอสกีได้เช่นกัน ตัวอย่างเช่น

```
-->ascii(91)
 ans  =
 [
-->ascii('[')
ans  =
    91.
```

7.1.4 คำสั่ง `fprintf`

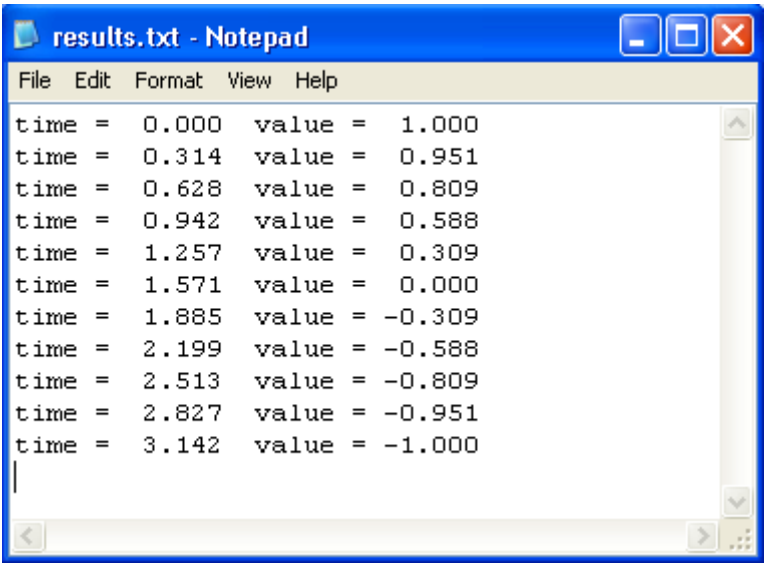
เป็นคำสั่งที่ทำหน้าที่คล้ายกับคำสั่ง `printf` แต่จะทำการพิมพ์ค่าของตัวแปรลงไปเก็บไว้ในไฟล์แทนที่จะแสดงผลออกมาที่หน้าต่างคำสั่ง รูปแบบการใช้งานของคำสั่งนี้คือ

```
fprintf(file, format, value_1, ..., value_n)
```

โดยที่พารามิเตอร์

- `file` เป็นชื่อของไฟล์ที่ต้องการจะให้เก็บค่าของตัวแปร `value_i`
- `format` และ `value_i` ทำหน้าที่เหมือนกันกับที่ใช้ในคำสั่ง `printf`

ตัวอย่างการใช้งานคำสั่งนี้ เช่น



รูปที่ 7.1 ใช้โปรแกรม Notepad ในการเรียกดูข้อมูลภายในไฟล์ `results.txt`

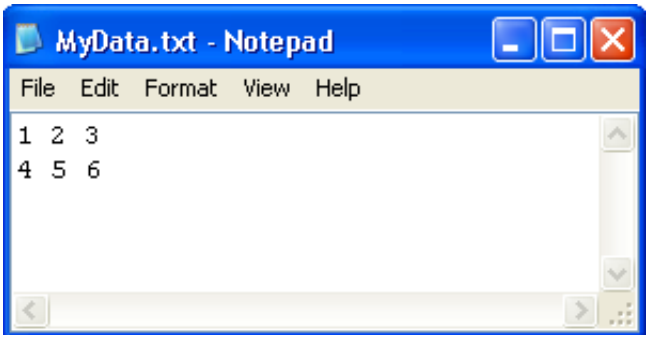
```
-->u = file('open','results.txt','unknown');
-->for t = 0:%pi/10:%pi
-->    fprintf(u,'time=%6.3f  value=%6.3f\n',t,cos(t));
-->end
-->file('close',u);
```

ผลลัพธ์ที่ได้คือโปรแกรม SCILAB จะทำการสร้างไฟล์ที่ชื่อว่า `results.txt` ในสารบบที่กำลังทำงานอยู่ซึ่งจะมีข้อมูลที่กำหนดโดยคำสั่ง `fprintf` ปรากฏอยู่ ถ้าลองใช้โปรแกรมเอดิเตอร์ เช่น Notepad หรือ WordPad เปิดไฟล์ `results.txt` ขึ้นมาดูก็จะพบข้อมูลตามที่กำหนดไว้ ดังแสดงในรูปที่ 7.1

7.1.5 คำสั่ง `scanf`

เป็นคำสั่งที่ใช้ในการรับค่าจากคีย์บอร์ดมาเก็บไว้ในตัวแปร ซึ่งมีรูปแบบการใช้งานดังนี้

```
-->printf('Enter x and y in the same line: X Y\n');
Enter x and y in the same line: X Y
```



รูปที่ 7.2 ข้อมูลภายในไฟล์ `MyData.txt`

```
-->[x,y] = scanf('%f  %f')

-->2.3 -3.3                                    //เว้นวรรคหนึ่งช่องไฟระหว่างข้อมูลแต่ละตัว
 y   =
  - 3.3
 x   =
    2.3
```

7.1.6 คำสั่ง `fscanf`

เป็นคำสั่งที่ใช้ในการอ่านค่าจากไฟล์ไปเก็บไว้ในตัวแปร โดยมีรูปแบบการใช้งานตามตัวอย่างที่จะแสดงต่อไปนี้ เริ่มต้นให้ทำการสร้างไฟล์ `MyData.txt` ในสารบบ `c:\` เพื่อเก็บข้อมูลตัวเลข 1 ถึง 6 ตามที่แสดงในรูปที่ 7.2 โดยใช้ชุดคำสั่งดังนี้

```
-->u = file('open', 'c:\MyData.txt', 'unknown');
-->fprintf(u, '1   2   3\n');
-->fprintf(u, '4   5   6\n');
-->file('close', u);
```

จากนั้นให้ทำการเปิดไฟล์ `MyData.txt` แล้วใช้คำสั่ง `fscanf` เพื่อทำการอ่านข้อมูลในไฟล์นี้มาเก็บไว้ในตัวแปร `a`, `b`, และ `c` โดยเมื่ออ่านข้อมูลจากไฟล์เสร็จแล้วก็ให้ทำการปิดไฟล์ ขั้นตอนที่กล่าวมานี้สามารถเขียนเป็นชุดคำสั่งได้ คือ

```
-->w = file('open', 'c:\MyData.txt', 'old');
-->aa = [];  bb = [];  cc = [];
-->for j = 1:2
-->  [a b c] = fscanf(w, '%f %f %f');
-->  aa = [aa; a];
-->  bb = [bb; b];
-->  cc = [cc; c];
-->end
-->file('close', w);
```

ซึ่งเมื่อทดลองเรียกดูข้อมูลของตัวแปร aa, bb, และ cc จะพบว่าได้ข้อมูลครบถ้วนเหมือนกับข้อมูลที่อยู่ในไฟล์ MyData.txt นั่นคือ

```
-->[aa bb cc]
 ans  =
    1.    2.    3.
    4.    5.    6.
```

7.1.7 คำสั่ง read

เป็นคำสั่งที่ใช้ในการอ่านข้อมูลจากไฟล์มาเก็บไว้ในตัวแปร ซึ่งมีรูปแบบการใช้งานดังนี้

```
[x] = read(file_desc, m, n, [format])
```

โดยที่พารามิเตอร์

- file_desc คือ ตัวแปรที่ใช้อ้างอิงถึงชื่อของไฟล์ที่ต้องการจะอ่านข้อมูล
- m คือ จำนวนแถวของข้อมูลในไฟล์ที่ต้องการจะอ่านข้อมูล (ใช้ m = -1 ถ้าต้องการให้อ่านข้อมูลทุกแถว)
- n คือ จำนวนแนวตั้งของข้อมูลในไฟล์ที่ต้องการจะอ่านข้อมูล
- format เป็นตัวเลือกที่กำหนดลักษณะการอ่านข้อมูลจากไฟล์ตามรูปแบบของภาษาฟอร์แทรน (FORTRAN) โดยมีรูปแบบดังนี้

  o iw โดยที่ i สำหรับเลขจำนวนเต็ม และ w คือจำนวนของตัวอักขระที่ยอมให้มีได้

- `fw.d` โดยที่ `f` สำหรับเลขจำนวนจริง, `w` คือจำนวนของตัวอักขระที่ยอมให้มีได้, และ `d` คือจำนวนของจุดทศนิยม (`d` ≥ `w+3`)
- `ew.d` โดยที่ `e` สำหรับเลขจำนวนจริงที่แสดงในรูปของเลขยกกำลัง, `w` คือจำนวนของตัวอักขระที่ยอมให้มีได้, และ `d` คือจำนวนของจุดทศนิยม (`d` ≥ `w+7`)
- `aw` โดยที่ `a` สำหรับสายอักขระ และ `w` คือจำนวนของตัวอักขระที่ยอมให้มีได้

สำหรับตัวอย่างการใช้งานคำสั่ง `read` สามารถดูได้ในหัวข้อที่ 7.2.1

7.1.8 คำสั่ง `write`

เป็นคำสั่งที่ใช้ในการเขียนข้อมูลลงไปเก็บไว้ในไฟล์ โดยมีรูปแบบการใช้งานดังนี้

```
write(file_desc, a, [format])
```

โดยที่พารามิเตอร์

- `file_desc` คือ ตัวแปรที่อ้างอิงถึงชื่อของไฟล์ที่ต้องการจะเขียนข้อมูล
- `a` คือ เวกเตอร์หรือเมทริกซ์ของสายอักขระที่จะนำไปบันทึกไว้ในไฟล์
- `format` เป็นตัวกำหนดลักษณะการเขียนข้อมูลลงไปในไฟล์ตามรูปแบบของภาษาฟอร์แทรนเหมือนกับพารามิเตอร์ `format` ที่ใช้ในคำสั่ง `read` ยกตัวอย่างเช่น ถ้าใช้ `'f10.3'` จะหมายถึงให้เขียนเป็นเลขจำนวนจริงสิบหลัก โดยมีเลขที่ตามหลังจุดทศนิยมเป็นจำนวนสามหลัก

สำหรับตัวอย่างการใช้งานคำสั่ง `write` สามารถดูได้ในหัวข้อที่ 7.2.2

7.2 ตัวอย่างการเขียนโปรแกรมเพื่อติดต่อระบบอินพุตและเอาต์พุต

ในส่วนนี้จะขอยกตัวอย่างการเขียนโปรแกรม เพื่อใช้ในการติดต่อระบบอินพุตและเอาต์พุตของโปรแกรม SCILAB เริ่มต้นจะอธิบายถึงตัวอย่างฟังก์ชันสำหรับใช้ในการคำนวณหาพื้นที่ของรูปสี่เหลี่ยม, สามเหลี่ยม, และวงกลม ดังต่อไปนี้

```
function [] = MyArea()

disp("============================")
disp("       Area calculation")
disp("============================")
disp("Select an option:")
disp(" 1 - Rectangular")
disp(" 2 - Triangular")
disp(" 3 - Circular")
disp("============================")

itype = input("")

select itype
  case 1 then
     id = "rectangular";
     w = input("Enter the width:")
     h = input("Enter the height:")
     A = w*h;
  case 2 then
     id = "triangular";
     b = input("Enter the width:")
     h = input("Enter the height:")
     A = b*h/2;
  case 3 then
     id = "circular";
     r = input("Enter the radius:")
     A = %pi*r^2;
  else
     error("Must choose only 1, 2 or 3    ")
end

printf("The area for a "+id+" cross-section is %10.6f .",A)

endfunction
```

เมื่อเขียนฟังก์ชันเสร็จแล้วก็ให้บันทึกลงในไฟล์ที่ชื่อว่า MyArea.sci จากนั้นทำการโหลดไฟล์นี้เพื่อเรียกใช้งานฟังก์ชัน MyArea โดยใช้คำสั่ง exec หรือ getf ตัวอย่างเช่น

```
-->getf('MyArea.sci')

-->MyArea
 ============================
        Area calculation
 ============================
 Select an option:
  1 - Rectangular
  2 - Triangular
  3 - Circular
 ============================
-->2
Enter the width:-->10
Enter the height:-->5
The area for a triangular cross-section is  25.000000 .

-->MyArea
 ============================
        Area calculation
 ============================
 Select an option:
  1 - Rectangular
  2 - Triangular
  3 - Circular
 ============================
-->4
!--error 10000
Must choose only 1, 2 or 3
at line      27 of function MyArea called by :
MyArea
```

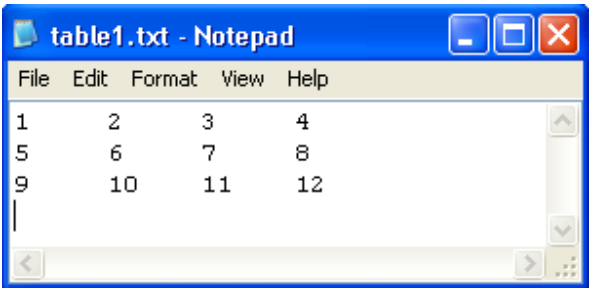
พิจารณาตัวอย่างต่อไปนี้ซึ่งเป็นฟังก์ชันสำหรับการสร้างตารางข้อมูล เพื่อที่จะได้เข้าใจถึงลักษณะการเขียนโปรแกรมเพื่อติดต่อระบบอินพุตและเอาต์พุตมากยิ่งขึ้น

```
function [] = MyTable()
printf("=============================================\n");
printf("    a          b           c              d    \n");
printf("=============================================\n");
for j = 1:8
   a = sin(2*j);
   b = cos(2*j);
   c = a + b;
   d = a - b;
   printf("%+6.5f  %+6.5f  %+6.5e  %+6.5e\n",a,b,c,d);
end
printf("=============================================\n");
endfunction
```

เมื่อเขียนฟังก์ชันเสร็จแล้วก็ให้บันทึกลงในไฟล์ที่ชื่อว่า `MyTable.sci` จากนั้นให้ทำการโหลดไฟล์นี้เพื่อเรียกใช้งานฟังก์ชัน `MyTable` โดยใช้คำสั่ง `exec` หรือ `getf` ตัวอย่างเช่น

```
-->getf("MyTable.sci")
-->MyTable
=============================================
    a          b           c              d
=============================================
+0.90930  -0.41615  +4.93151e-001  +1.32544e+000
-0.75680  -0.65364  -1.41045e+000  -1.03159e-001
-0.27942  +0.96017  +6.80755e-001  -1.23959e+000
+0.98936  -0.14550  +8.43858e-001  +1.13486e+000
-0.54402  -0.83907  -1.38309e+000  +2.95050e-001
-0.53657  +0.84385  +3.07281e-001  -1.38043e+000
+0.99061  +0.13674  +1.12734e+000  +8.53870e-001
-0.28790  -0.95766  -1.24556e+000  +6.69756e-001
=============================================
```

สังเกตจะพบว่าในฟังก์ชัน `MyTable` มีการใช้รหัสบังคับการพิมพ์ `%+6.5f` เพื่อเป็นการบอกว่าให้แสดงเครื่องหมายบวกหรือลบหน้าผลลัพธ์ด้วย ส่วนรหัสบังคับการพิมพ์ `%+6.5e` หมายถึงให้แสดงผลลัพธ์ในรูปแบบของเลขยกกำลัง



รูปที่ 7.3 ข้อมูลภายในไฟล์ `table1.txt`

7.2.1 การอ่านข้อมูลจากไฟล์

ในส่วนนี้จะอธิบายตัวอย่างการใช้งานคำสั่ง `read` เพื่ออ่านข้อมูลจากไฟล์มาเก็บไว้ในตัวแปร สมมติว่ามีไฟล์ที่ชื่อ `table1.txt` ในสารบบ `c:\` ซึ่งมีข้อมูลภายในไฟล์ตามรูปที่ 7.3 ถ้าต้องการสร้างฟังก์ชันเพื่อนำข้อมูลแต่ละแนวตั้งในไฟล์นี้มาบรรจุไว้ในตัวแปร `x1`, `x2`, `x3`, และ `x4` ก็สามารถทำได้ดังนี้

```
function [x1, x2, x3, x4] = MyReadTable1()
//|-------------------------------------------------|
//| The name of a file contains a table.            |
//| This function reads the table out of the file.  |
//|-------------------------------------------------|
printf("Reading a table from a file\n");
printf("===========================\n");
filename = input("Enter filename between quotes:\n");
u = file('open', filename, 'old');
Table = read(u, -1, 4);          //ไม่รู้ว่ามีข้อมูลกี่แถว แต่รู้ว่ามีข้อมูล 4 แนวตั้ง
[n, m] = size(Table);
printf("There are %d columns in the table\n", m);
for j = 1:m
   execstr('x' + string(j) + ' = Table(:,j)');
end
file('close', u);
endfunction
```

เมื่อเขียนฟังก์ชันเสร็จแล้วก็ให้บันทึกลงในไฟล์ที่ชื่อว่า MyReadTable1.sci จากนั้นทำการโหลดไฟล์นี้เพื่อเรียกใช้งานฟังก์ชัน MyReadTable1 โดยใช้คำสั่ง exec หรือ getf ตัวอย่างเช่น

```
-->getf('MyReadTable1.sci')
-->[x1, x2, x3, x4] = MyReadTable1()
Reading a table from a file
============================
 Enter filename between quotes:
-->'c:\table1.txt'
 There are 4 columns in the table
 x4  =
    4.
    8.
    12.
 x3  =
    3.
    7.
    11.
 x2  =
    2.
    6.
    10.
 x1  =
    1.
    5.
    9.
```

หมายเหตุ คำสั่ง Table = read(u, -1, 4) หมายถึงให้ทำการอ่านข้อมูลในไฟล์ที่กำหนดโดยพารามิเตอร์ u ส่วนอินพุตอาร์กิวเมนต์ตัวที่สองที่มีค่าเท่ากับ -1 หมายถึงให้โปรแกรม SCILAB อ่านข้อมูลทุกแถวในไฟล์ ส่วนอินพุตอาร์กิวเมนต์ตัวที่สามที่มีค่าเท่ากับ 4 หมายถึงให้โปรแกรม SCILAB อ่านข้อมูลเป็นจำนวนห้าแนวตั้งในไฟล์ สำหรับคำสั่ง execstr เป็นคำสั่งที่บอกให้โปรแกรม SCILAB ทำการประมวลผลค่าของสายอักขระ

นอกจากการใช้คำสั่ง read เพื่ออ่านข้อมูลจากไฟล์แล้ว ผู้ใช้ยังสามารถใช้คำสั่ง fscanf ได้เช่นกันซึ่งมีรูปแบบการใช้งานดังนี้

```
function [aa, bb, cc, dd] = MyReadTable2()
//|-----------------------------------------------|
//| This function is similar to MyReadTable1() but |
//| using fscanf() instead of read() to read data. |
//|-----------------------------------------------|
filename = input("Enter filename between quotes:\n");
u = file('open', filename, 'old');
aa = []; bb = []; cc = []; dd = [];
for j = 1:3
   [a, b, c, d] = fscanf(u,'%f  %f  %f  %f');
   aa = [aa; a];
   bb = [bb; b];
   cc = [cc; c];
   dd = [dd; d];
end
file('close', u);
endfunction
```

เมื่อเขียนฟังก์ชันเสร็จแล้วก็ให้บันทึกลงในไฟล์ที่ชื่อว่า MyReadTable2.sci จากนั้นทำการโหลดไฟล์นี้เพื่อเรียกใช้งานฟังก์ชัน MyReadTable2 โดยใช้คำสั่ง exec หรือ getf ตัวอย่างเช่น

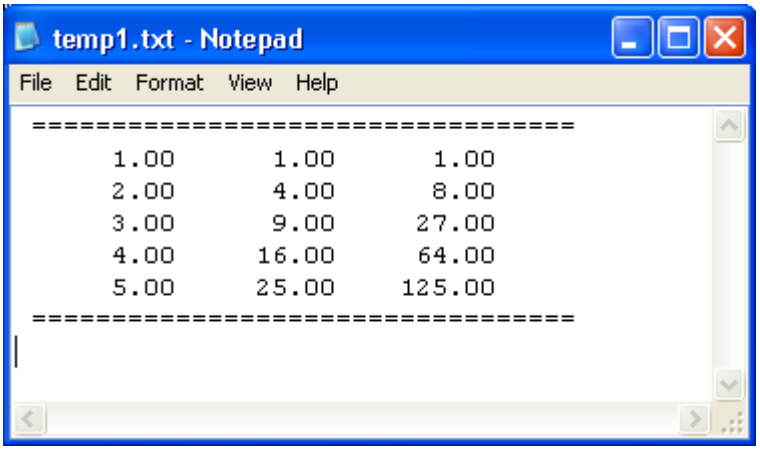
```
-->getf('MyReadTable2.sci')
-->[aa, bb, cc, dd] = MyReadTable2()
Enter filename between quotes:
-->'c:\table1.txt'          //สมมติว่าไฟล์นี้อยู่ในสารบบ c:\ และมีข้อมูลตามรูปที่ 7.3
 dd  =
    4.
    8.
    12.
 cc  =
    3.
    7.
    11.
```

```
bb  =
   2.
   6.
   10.
aa  =
   1.
   5.
   9.
```

7.2.2 การเขียนข้อมูลลงในไฟล์

ในส่วนนี้จะขอยกตัวอย่างการใช้งานคำสั่ง `write` เพื่อเขียนข้อมูลเข้าไปเก็บไว้ในไฟล์ดังต่อไปนี้ สมมติว่าต้องการสร้างฟังก์ชันชื่อ `MyWriteToFile1.sci` เพื่อบันทึกค่าของตัวแปร x, x^2, และ x^3 สำหรับ x = 1, 2, 3, 4, และ 5 ลงไปเก็บไว้ในไฟล์ชื่อ `temp1.txt` ในสารบบ `c:\` ก็สามารถทำได้ดังนี้

```
function [] = MyWriteToFile1()

//|-------------------------------------------|
//|  This function write the data into a file.  |
//|-------------------------------------------|

printf("Printing to a file\n");
printf("======================\n");
filename = input("Enter file to write to (between quotes):\n");
u = file('open', filename, 'new');
write(u, "==================================");

for j = 1:5
  write(u, [j  j*j  j^3], '(f10.2, f10.2, f10.2)');
end

write(u, " ==================================");
file('close',u)

endfunction
```



รูปที่ 7.4 ข้อมูลภายในไฟล์ `temp1.txt`

เมื่อเขียนฟังก์ชันเสร็จแล้วก็ให้บันทึกลงในไฟล์ที่ชื่อว่า `MyWriteToFile1.sci` จากนั้นทำการโหลดไฟล์นี้เพื่อเรียกใช้งานฟังก์ชัน `MyWriteToFile1` โดยใช้คำสั่ง `exec` หรือ `getf` ตัวอย่างเช่น

```
-->getf('MyWriteToFile1.sci')
-->MyWriteToFile1
Printing to a file
==================
Enter file to write to (between quotes):
-->'c:\temp1.txt'
-->
```

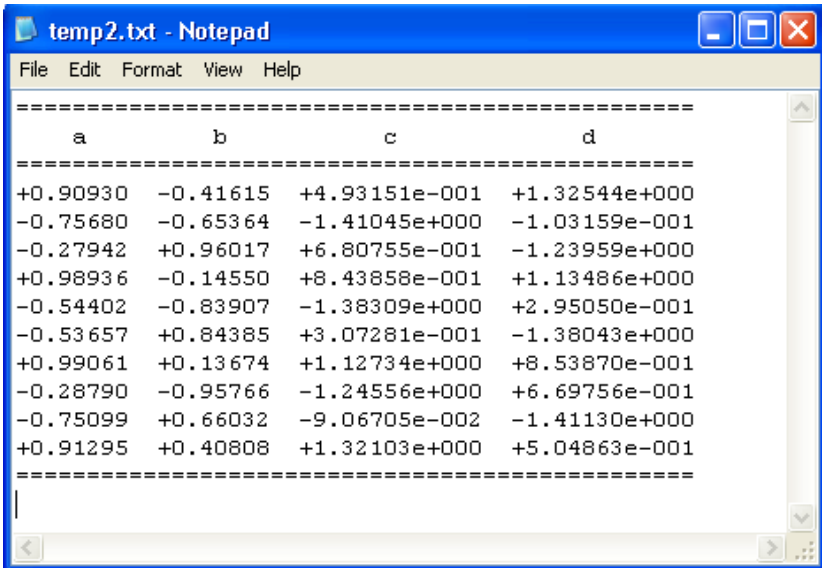
ผลลัพธ์ที่ได้คือข้อมูลที่ต้องการจะถูกเขียนเข้าไปเก็บไว้ในไฟล์ชื่อ `temp1.txt` ซึ่งเมื่อใช้โปรแกรม Notepad เปิดไฟล์นี้ก็จะได้ตามรูปที่ 7.4

นอกจากการใช้คำสั่ง `write` เพื่อเขียนข้อมูลลงไปเก็บไว้ในไฟล์แล้ว ผู้ใช้ยังสามารถใช้คำสั่ง `fprintf` ได้เช่นกัน โดยมีรูปแบบการเรียกใช้งานที่แตกต่างกันเล็กน้อยดังแสดงในตัวอย่างต่อไปนี้ (ศึกษารายละเอียดการใช้งานคำสั่ง `write` และ `fprintf` ได้จากคำสั่ง `help`)

```
function [] = MyWriteToFile2()
//|-------------------------------------------|
//|  This function write the data into a file.  |
//|-------------------------------------------|
printf("Printing to a file\n");
printf("==================\n");
filname = input("Enter file to write (between quotes):\n");
u = file('open', filname, 'new');
printf("====================================\n");
fprintf(u, "====================================\n");
printf("    a         b           c              d  \n");
fprintf(u, "    a         b           c              d   \n");
printf("====================================\n");
fprintf(u, "====================================\n");
for j = 1:10
  a = sin(2*j);
  b = cos(2*j);
  c = a + b;
  d = a - b;
  printf("%+6.5f  %+6.5f  %+6.5e  %+6.5e\n", a, b, c, d);
  fprintf(u,"%+6.5f  %+6.5f  %+6.5e  %+6.5e\n", a, b, c, d);
end
printf("====================================\n");
fprintf(u, "====================================\n");
file('close', u)
endfunction
```

เมื่อเขียนฟังก์ชันเสร็จแล้วก็ให้บันทึกลงในไฟล์ที่ชื่อว่า `MyWriteToFile2.sci` จากนั้นทำการโหลดไฟล์นี้เพื่อเรียกใช้งานฟังก์ชัน `MyWriteToFile2` โดยใช้คำสั่ง `exec` หรือ `getf` ตัวอย่างเช่น

```
-->getf('MyWriteToFile2.sci')
-->MyWriteToFile2
```



รูปที่ 7.5 ข้อมูลภายในไฟล์ `temp2.txt`

```
Printing to a file
==================
Enter file to write (between quotes):
-->'c:\temp2.txt'
================================================
    a         b           c              d
================================================
+0.90930  -0.41615  +4.93151e-001  +1.32544e+000
-0.75680  -0.65364  -1.41045e+000  -1.03159e-001
-0.27942  +0.96017  +6.80755e-001  -1.23959e+000
+0.98936  -0.14550  +8.43858e-001  +1.13486e+000
-0.54402  -0.83907  -1.38309e+000  +2.95050e-001
-0.53657  +0.84385  +3.07281e-001  -1.38043e+000
+0.99061  +0.13674  +1.12734e+000  +8.53870e-001
-0.28790  -0.95766  -1.24556e+000  +6.69756e-001
-0.75099  +0.66032  -9.06705e-002  -1.41130e+000
+0.91295  +0.40808  +1.32103e+000  +5.04863e-001
================================================
```

ซึ่งผลลัพธ์ที่ได้นี้จะถูกนำไปเก็บไว้ในไฟล์ที่ชื่อว่า `temp2.txt` ในสารบบ `c:\` ดังนั้นถ้าเรียกดูข้อมูลในไฟล์นี้โดยใช้โปรแกรม Notepad ก็จะได้ผลลัพธ์ตามรูปที่ 7.5

7.3 สรุป

ในบทนี้ได้อธิบายถึงรูปแบบการใช้งานคำสั่งต่างๆ เช่น คำสั่ง `input`, `file`, `read`, `write`, `printf`, และ `scanf` เป็นต้น เพื่อใช้ในการติดต่อระบบอินพุตและเอาต์พุตซึ่งเป็นระบบที่เกี่ยวข้องกับการอ่านและเขียนข้อมูลระหว่างโปรแกรม SCILAB กับเครื่องคอมพิวเตอร์ กล่าวคือ ผู้ใช้สามารถที่จะอ่านข้อมูลจากไฟล์มาเก็บไว้ในตัวแปรที่ใช้ภายในโปรแกรม SCILAB หรือเขียนข้อมูลค่าของตัวแปรจากโปรแกรม SCILAB เข้าไปเก็บไว้ในไฟล์ก็ได้ ดังนั้นเมื่อเข้าใจถึงลักษณะการใช้งานคำสั่งต่างๆ ที่ใช้ในการติดต่อระบบอินพุตและเอาต์พุตของโปรแกรม SCILAB แล้ว ก็จะทำให้สามารถออกแบบและพัฒนาฟังก์ชันใหม่ๆ ให้มีความสามารถในการโต้ตอบกันระหว่างตัวโปรแกรม SCILAB กับผู้พัฒนาโปรแกรมได้

7.4 แบบฝึกหัดท้ายบท

7.1 จงเขียนโปรแกรมเพื่อหาผลลัพธ์ของ การบวก การลบ การคูณ และการหาร ของเลขสองจำนวน `x` และ `y` โดยให้ตัวโปรแกรมรับค่าของตัวแปร `x` และ `y` จากคีย์บอร์ด โดยใช้คำสั่ง `input` และแสดงผลลัพธ์ที่ได้ออกมาที่หน้าต่างคำสั่ง ผ่านทางคำสั่ง `printf`

7.2 ทำเหมือนกับข้อ 7.1 แต่ใช้คำสั่ง `scanf` ในการรับค่าของตัวแปร `x` และ `y` จากคีย์บอร์ด

7.3 กำหนดให้ข้อมูลภายในไฟล์ชื่อ `test1.txt` มีดังนี้

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40

7.3.1) จงเขียนโปรแกรมเพื่ออ่านข้อมูลจากไฟล์นี้มาเก็บไว้ตัวแปร `x` โดยใช้คำสั่ง `read` แล้วสั่งให้แสดงผลลัพธ์ค่าของตัวแปร `x` ออกทางหน้าต่างคำสั่ง

7.3.2) ทำเหมือนกับข้อ 7.3.1 แต่ให้อ่านข้อมูลจากไฟล์โดยใช้คำสั่ง `fscanf`

7.3.3) เปรียบเทียบผลลัพธ์ที่ได้จากข้อ 7.3.1 และ 7.3.2

7.4 กำหนดให้ตัวแปร `t = -5:0.2:5`

7.4.1) จงเขียนโปรแกรมเพื่อคำนวณหาค่า `y = sin(t)/t` โดยให้แสดงผลลัพธ์ของค่า `y` ออกทางหน้าต่างคำสั่ง พร้อมทั้งเขียนข้อมูลค่าของตัวแปร `y` เข้าไปเก็บไว้ในไฟล์ชื่อ `test2.txt` โดยใช้คำสั่ง `write`

7.4.2) ทำเหมือนกับข้อ 7.4.1 แต่ให้เขียนข้อมูลเข้าไปเก็บในไฟล์ชื่อ `test3.txt` โดยใช้คำสั่ง `fprintf`

7.4.3) เปรียบเทียบผลลัพธ์ที่ได้จากข้อ 7.4.1 และ 7.4.2

7.5 จงเปรียบเทียบข้อแตกต่างของคำสั่ง `scanf`, `fscanf`, และ `read`

7.6 จงเปรียบเทียบข้อแตกต่างของคำสั่ง `printf`, `fprintf`, และ `write`

บทที่ 8

การตรวจสอบหาข้อผิดพลาดของโปรแกรม

การพัฒนาโปรแกรมโดยทั่วไป สิ่งที่ผู้พัฒนาโปรแกรมจะต้องพบเสมอก็คือ โปรแกรมที่พัฒนาขึ้นมาไม่สามารถทำงานได้ตามที่คาดหวังไว้อันเนื่องมาจากมีจุดบกพร่อง (bug) ภายในตัวโปรแกรมซึ่งอาจจะเกิดจากสาเหตุต่างๆ เช่น ข้อผิดพลาดทางด้านไวยากรณ์ (syntax error) และข้อผิดพลาดจากอัลกอริทึม เป็นต้น ข้อผิดพลาดเหล่านี้จะทำให้โปรแกรมที่พัฒนาขึ้นมาไม่สามารถทำงานได้หรือถ้าทำงานได้แต่ก็จะให้ผลลัพธ์ผิดไปจากที่คาดหวังไว้ ดังนั้นผู้พัฒนาโปรแกรมมีความจำเป็นที่จะต้องเข้าใจเกี่ยวกับหลักการทำงานของฟังก์ชันทั้งหมดที่ใช้ภายในโปรแกรมอย่างถูกต้อง หรือถ้ามีข้อผิดพลาดเกิดขึ้น ผู้พัฒนาโปรแกรมก็ควรที่จะสามารถค้นหาได้ว่าอะไรเป็นสาเหตุของข้อผิดพลาดเหล่านั้น (หรือสามารถหาตำแหน่งของจุดบกพร่องได้) เพื่อที่จะได้ทำการแก้ไขตัวโปรแกรมให้ถูกต้อง ซึ่งขั้นตอนนี้จะเรียกว่าการแก้จุดบกพร่อง (debugging) ดังนั้นในบทนี้จะอธิบายถึงขั้นตอนและคำสั่งต่างๆ ที่ใช้ในการตรวจสอบหาข้อผิดพลาดของโปรแกรม

8.1 ขั้นตอนการตรวจสอบหาข้อผิดพลาด

ในส่วนนี้จะอธิบายถึงขั้นตอนและคำสั่งต่างๆ ที่ใช้ในการตรวจสอบหาข้อผิดพลาดของโปรแกรมที่พัฒนาขึ้นมา ให้พิจารณาสมการพหุนาม $ax^2 + bx + c = 0$ ซึ่งคำตอบของสมการนี้มีสองคำตอบ คือ $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ ตัวอย่างเช่น ถ้ากำหนดให้ $x^2 - 3x + 2 = 0$ จะได้ว่าคำตอบของสมการนี้คือ $x = 1$ และ $x = 2$ ผู้ใช้สามารถเขียนโปรแกรมเพื่อคำนวณหาคำตอบของสมการพหุนามนี้ได้ดังแสดงในรูปที่ 8.1

```
function [x1, x2] = MyRoot(a, b, c)                    //บรรทัดที่ 1
num1 = -b + sqrt(b^2 - 4*a*c);                         //บรรทัดที่ 2
num2 = -b - sqrt(b^2 - 4*a*c);                         //บรรทัดที่ 3
den  = 2*a;                                            //บรรทัดที่ 4
x1   = num1/den;                                       //บรรทัดที่ 5
x2   = num2/den;                                       //บรรทัดที่ 6
endfunction                                            //บรรทัดที่ 7
```

รูปที่ 8.1 รายละเอียดชุดคำสั่งภายในไฟล์ฟังก์ชัน `MyRoot.sci`

เมื่อเขียนโปรแกรมเสร็จแล้วก็ทำการบันทึกฟังก์ชันนี้ลงในไฟล์ที่ชื่อว่า `MyRoot.sci` จากนั้นให้ทำการโหลดชื่อไฟล์นี้เพื่อเรียกใช้งานฟังก์ชันนี้โดยใช้คำสั่ง `exec` หรือ `getf` ดังนี้

```
-->getf('MyRoot.sci');

-->a = 1;  b = -3;  c = 2;

-->[x1, x2] = MyRoot(a, b, c)
 x2  =
    1.
 x1  =
    2.
```

ซึ่งจะได้ผลลัพธ์ถูกต้องตามที่ต้องการ

ในการหาข้อผิดพลาดภายในตัวโปรแกรม ขั้นตอนแรกคือผู้ใช้จะต้องทำการขัดจังหวะ (interruption) การประมวลผลของโปรแกรมเพื่อตรวจสอบดูค่าของตัวแปรต่างๆ ณ เวลาขณะนั้นว่ามีค่าเป็นไปตามที่คาดหวังไว้จากตัวโปรแกรมหรือไม่ ถ้าผลลัพธ์ที่ได้ผิดไปจากที่คาดหวังไว้แสดงว่าอาจจะมีข้อผิดพลาดเกิดขึ้นภายในตัวโปรแกรม**ก่อน**จะถึงตำแหน่งที่ทำการขัดจังหวะ แต่ถ้าหากผลลัพธ์ที่ได้เป็นไปตามที่คาดหวังไว้ก็แสดงว่า ไม่มีข้อผิดพลาดภายในตัวโปรแกรมก่อนที่จะถึงตำแหน่งที่ทำการขัดจังหวะ ในทางปฏิบัติการขัดจังหวะโปรแกรมสามารถทำได้ 3 วิธี คือ

1) กดปุ่ม Ctrl-C จากคีย์บอร์ด
2) ใช้คำสั่ง `setbpt` (มาจากคำว่า `set breakpoint`) ที่หน้าต่างคำสั่ง
3) ใช้คำสั่ง `pause` ใส่เข้าไปในตัวโปรแกรม ก่อนจะทำการประมวลผล

ข้อแตกต่างที่เห็นได้ชัดระหว่างคำสั่ง `setbpt` และ `pause` คือคำสั่ง `setbpt` จะไม่ไปเปลี่ยนแปลงตัวโปรแกรมที่พัฒนาขึ้นมา ในขณะที่คำสั่ง `pause` จะไปเปลี่ยนแปลงตัวโปรแกรม เนื่องจากคำสั่ง `pause` จะต้องถูกใส่เข้าไปในตัวโปรแกรมนั้นก่อนจะทำการประมวลผล

8.1.1 คำสั่ง `setbpt`

คำสั่ง `setbpt` เป็นคำสั่งที่ใช้ในการกำหนดตำแหน่งของจุดพัก[26] (breakpoint) ว่าจะให้อยู่ที่บรรทัดใดในตัวโปรแกรมเพื่อที่ว่าเวลาประมวลผลโปรแกรมนั้นแล้ว ตัวโปรแกรมจะมาหยุดแบบชั่วคราว ณ บรรทัดที่มีจุดพัก เพื่อให้ผู้ใช้สามารถเปลี่ยนแปลงค่าพารามิเตอร์หรือตรวจสอบความถูกต้องของตัวโปรแกรมตั้งแต่บรรทัดแรก (หรือบรรทัดที่มีจุดพักก่อนหน้านี้) จนถึงบรรทัด**ก่อน**ที่จะมีจุดพักนั้นได้ ให้พิจารณาชุดคำสั่งต่อไปนี้จะได้เข้าใจถึงหลักการใช้งานคำสั่ง `setbpt`

```
-->clear;
-->getf('MyRoot.sci');
-->a = 1;  b = -3;  c = 2;
-->setbpt('MyRoot', 5)        //บรรทัดที่ห้าในโปรแกรม MyRoot จะยังไม่ถูกนำไปประมวลผล
```

ชุดคำสั่งนี้จะทำการลบข้อมูลทั้งหมดในหน้าต่างคำสั่งโดยใช้คำสั่ง `clear` จากนั้นก็ทำการโหลดไฟล์ `MyRoot.sci` เพื่อเรียกใช้งานโปรแกรม `MyRoot` พร้อมทั้งกำหนดค่าของตัวแปร `a`, `b`, และ `c` ส่วนคำสั่งสุดท้ายเป็นการกำหนดตำแหน่งของจุดพักให้อยู่ในบรรทัดที่ห้าภายในโปรแกรม `MyRoot` หลังจากนั้นถ้าทำการประมวลผลโปรแกรม `MyRoot` จะได้ผลดังนี้

```
-->[x1, x2] = MyRoot(a, b, c)
Stop after row     5 in function MyRoot :
-1->                                   //เครื่องหมายแสดงระดับการขัดจังหวะ ณ ระดับที่ 1
```

ผลลัพธ์ที่ได้คือโปรแกรม `MyRoot` จะถูกขัดจังหวะแบบชั่วคราว ณ บรรทัดที่ห้าของตัวโปรแกรม `MyRoot` (นั่นคือบรรทัดที่ห้าจะยังไม่ถูกนำไปประมวลผล) โดยจะแสดงเครื่องหมาย Interruption

[26] คำสั่งในบรรทัดที่มีจุดพัก จะยังไม่ถูกนำไปประมวลผล

prompt[27] "-1->" เพื่อเป็นการบอกว่าตอนนี้โปรแกรม MyRoot ได้อยู่ที่ระดับที่ 1 ของการขัดจังหวะ ซึ่ง ณ ตำแหน่งนี้ผู้ใช้สามารถตรวจสอบความถูกต้องของข้อมูลตั้งแต่บรรทัดที่หนึ่งจนถึงบรรทัดที่สี่ได้ ตัวอย่างเช่น ผู้ใช้สามารถเรียกดูค่าพารามิเตอร์ต่างๆ ที่ได้มีการกำหนดภายในโปรแกรม MyRoot ตั้งแต่บรรทัดที่หนึ่งจนถึงบรรทัดที่สี่ ซึ่งถ้าพารามิเตอร์เหล่านี้มีค่าตรงตามที่คาดหวังไว้ก็แสดงว่าคำสั่งทั้งหมดตั้งแต่บรรทัดที่หนึ่งจนถึงบรรทัดที่สี่มีความถูกต้อง เช่น

```
-1->[num1  num2  den]
 ans  =
    4.    2.    2.

-1->[x1  x2]
[x1  x2]
    --error     4
undefined variable : x1      //ตัวแปร x1 อยู่ในคำสั่งบรรทัดที่ห้าซึ่งยังไม่ได้ถูกนำไปประมวลผล
```

ผลลัพธ์ที่ได้แสดงให้เห็นว่าผู้ใช้สามารถเรียกดูค่าของพารามิเตอร์ num1, num2, และ den ได้และมีผลลัพธ์ตามที่คาดหวังไว้ อย่างไรก็ตาม ณ จุดนี้ผู้ใช้ยังไม่สามารถเรียกดูค่าของพารามิเตอร์ x1 และ x2 ได้เนื่องจากโปรแกรม MyRoot ยังไม่ได้ทำการประมวลผลคำสั่งในบรรทัดที่ห้าและบรรทัดหก (ดูรูปที่ 8.1)

8.1.2 คำสั่ง resume และ return

จากตัวอย่างในหัวข้อ 8.1.1 ถ้าต้องการให้โปรแกรม MyRoot ทำงานต่อไปจนจบโปรแกรม ก็สามารถทำได้โดยใช้คำสั่ง resume[28] ดังแสดงต่อไปนี้

---

[27] ในหนึ่งโปรแกรมสามารถมีระดับของการขัดจังหวะ (level of interruption) ได้หลายระดับ ถ้าภายในตัวโปรแกรมนั้นมีจุดพัก (breakpoint) มากกว่าหนึ่งจุด นอกจากนี้ผู้ใช้ยังสามารถเรียกดูค่าของพารามิเตอร์ต่างๆ ที่อยู่ในระดับของการขัดจังหวะที่ระดับต่ำกว่าได้ทั้งหมด

หมายเหตุ ขณะที่อยู่ที่ระดับของการขัดจังหวะ ณ ระดับหนึ่ง การทำงานโดยใช้คำสั่งต่างๆ ที่ระดับของการขัดจังหวะนี้จะไม่มีผลกระทบต่อค่าของตัวแปรใดๆ ที่อยู่ในระดับของการขัดจังหวะที่ระดับต่ำกว่า

[28] ในกรณีที่โปรแกรมมีระดับของการขัดจังหวะหลายๆ ระดับ คำสั่ง resume จะทำหน้าที่ในการลดระดับของการขัดจังหวะได้เช่นกัน

```
-1->resume
 x2  =                          //นั่นคือ x2 = num2/den
    1.
 x1  =                          //นั่นคือ x1 = num1/den
    2.
```

ผลลัพธ์ที่ได้คือค่าของตัวแปร x1 และ x2 เป็นคำตอบของสมการ $x^2 - 3x + 2 = 0$ ตามที่ต้องการ

นอกจากนี้ในขณะที่อยู่ ณ ระดับของการขัดจังหวะใดๆ ผู้ใช้ยังสามารถที่จะเปลี่ยนแปลงค่าของตัวแปรต่างๆ ได้ โดยที่หลังจากกำหนดค่าใหม่ให้กับตัวแปรแล้ว ถ้าใช้คำสั่ง resume ในลักษณะเดิมก็จะทำให้ได้ผลลัพธ์เหมือนเดิม ดังแสดงในตัวอย่างต่อไปนี้

```
-->setbpt('MyRoot', 5)
-->[x1, x2] = MyRoot(a,b,c)
Stop after row     5 in function MyRoot :
-1->[num1 num2 den]            //ค่าเดิมของพารามิเตอร์ต่างๆ เมื่อ a = 1, b = -3 และ c = 2
 ans  =
    4.    2.    2.
-1->num1=3;  num2=1;           //กำหนดค่าใหม่ให้กับพารามิเตอร์ num1 และ num2
-1->resume                     //ใช้คำสั่ง resume ในลักษณะเดิม จะทำให้ได้ผลลัพธ์เหมือนเดิม
 x2  =
    1.
 x1  =
    2.
```

หากต้องการให้โปรแกรม MyRoot นำค่าของตัวแปรที่ถูกกำหนดขึ้นมาใหม่นี้มาใช้ในคำนวณหาค่า x1 และ x2 แล้ว ผู้ใช้จะต้องใช้คำสั่ง resume ในลักษณะต่อไปนี้

```
-1->num1 = 3; num2 = 1;
-1->[num1, num2] = resume(num1, num2)
 x2  =                          //เนื่องจาก x2 = num2/den = 1/2 = 0.5
    0.5
```

```
 x1   =                          //เนื่องจาก x1 = num1/den = 3/2 = 1.5
    1.5
```

ผลลัพธ์ที่ได้จะเป็นคำตอบที่สอดคล้องกับค่าของ num1 และ num2 ที่ถูกกำหนดค่าให้ใหม่

หมายเหตุ คำสั่ง return จะทำงานได้เหมือนกับคำสั่ง resume ทุกประการ

8.1.3 คำสั่ง pause

คำสั่ง pause มีผลลัพธ์แบบเดียวกันกับการใช้คำสั่ง setbpt เพียงแต่จะต้องนำคำสั่ง pause ไปใส่ไว้ในตัวโปรแกรม**ก่อน**บรรทัดที่มีการกำหนดจุดพักด้วยคำสั่ง setbpt ยกตัวอย่างเช่น ถ้าใช้คำสั่ง

```
-->setbpt('MyRoot', 5)
```

จะมีผลลัพธ์เช่นเดียวกันกับการนำคำสั่ง pause ไปใส่ไว้ในตัวโปรแกรม MyRoot ก่อนคำสั่งในบรรทัดที่ห้าในรูปที่ 8.1

8.1.4 คำสั่งที่น่าสนใจ

ในส่วนนี้จะอธิบายคำสั่งที่น่าสนใจที่ใช้ในการตรวจสอบและแก้ไขโปรแกรมที่พัฒนาขึ้นมาดังนี้

8.1.4.1 คำสั่ง abort

เป็นคำสั่งที่ทำให้ระดับของการขัดจังหวะลดลงมาอยู่ที่ระดับที่ศูนย์ นั่นคือมาอยู่ที่เครื่องหมาย Scilab prompt “-->” ซึ่งเป็นระดับปรกติของหน้าต่างคำสั่งในโปรแกรม SCILAB

8.1.4.2 คำสั่ง dispbpt

เป็นคำสั่งที่ใช้แสดงจุดพักทั้งหมดที่มีอยู่ในตัวโปรแกรมว่าอยู่ที่บรรทัดใดบ้าง ตัวอย่างเช่น

```
-->setbpt('MyRoot', 5)

-->dispbpt
breakpoints of  function :MyRoot
         5
```

```
function [x1, x2] = MyRoot2(a, b, c)                    //บรรทัดที่ 1
num1 = -b + sqrt(b^2 - 4*a*c);                          //บรรทัดที่ 2
num2 = -b - sqrt(b^2 - 4*a*c);                          //บรรทัดที่ 3
den  = 2*a;                                             //บรรทัดที่ 4
pause;                                                  //บรรทัดที่ 5
x1   = num1/den;                                        //บรรทัดที่ 6
x2   = num2/den;                                        //บรรทัดที่ 7
endfunction                                             //บรรทัดที่ 8
```

รูปที่ 8.2 รายละเอียดชุดคำสั่งภายในโปรแกรม `MyRoot2.sci`

8.1.4.3 คำสั่ง `delbpt`

เป็นคำสั่งที่ใช้ในการลบจุดพักออกจากตัวโปรแกรม ตัวอย่างเช่น

```
-->delbpt('MyRoot', 5)
-->dispbpt
-->
```

จะเห็นได้ว่าหลังจากลบจุดพักแล้ว เมื่อใช้คำสั่ง `dispbpt` ก็จะไม่พบอะไร

8.1.4.4 คำสั่ง `whereami`

ในระหว่างการประมวลผลของตัวโปรแกรม ถ้ากดปุ่ม Ctrl-C หรือมีคำสั่ง `pause` อยู่ภายในตัวโปรแกรมเพื่อทำหน้าที่ขัดจังหวะการประมวลผลของโปรแกรม ผู้ใช้สามารถที่จะหาตำแหน่งของการขัดจังหวะได้โดยใช้คำสั่ง `whereami` ดังแสดงในตัวอย่างต่อไปนี้

ให้พิจารณาฟังก์ชัน `MyRoot2.sci` ตามรูปที่ 8.2 ซึ่งมีคำสั่ง `pause` อยู่ในบรรทัดที่ห้า เมื่อเรียกใช้งานโปรแกรมนี้จะได้ผลลัพธ์คือ

```
-->exec('MyRoot2.sci');
-->a = 1;   b = -3;   c = 2;
```

```
-->[x1, x2] = MyRoot2(a, b, c)
-1->
```

จะพบว่าโปรแกรม MyRoot2 ถูกขัดจังหวะการประมวลผลแต่ไม่รู้ว่าถูกขัดจังหวะ ณ บรรทัดใด ในกรณีนี้ถ้าต้องการทราบว่าโปรแกรม MyRoot2 ถูกขัดจังหวะที่บรรทัดใดก็สามารถทำได้โดยใช้คำสั่ง whereami ดังแสดงต่อไปนี้

```
-1->whereami
whereami called under pause
pause    called at line 6 of macro MyRoot2
-1->[num1  num2  den]
 ans  =
    4.    2.    2.
-1->[x1  x2]
[x1 x2]
   !--error 4
undefined variable : x1
```

ผลลัพธ์ที่ได้แสดงให้ทราบว่าโปรแกรม MyRoot2 ถูกขัดจังหวะโดยคำสั่ง pause เมื่อประมวลผลมาจนถึงบรรทัดที่หก (คำสั่งบรรทัดที่หกยังไม่ถูกนำไปประมวลผล) ดังนั้นจึงสามารถเรียกดูค่าของตัวแปร num1, num2, และ den ได้ แต่ไม่สามารถเรียกดูค่าของตัวแปร x1 และ x2

8.1.4.5 คำสั่ง where

คำสั่งนี้ทำหน้าที่คล้ายกับคำสั่ง whereami เพียงแต่จะบอกรายละเอียดทั้งหมดในรูปของตัวแปรเวกเตอร์ ดังแสดงในตัวอย่างต่อไปนี้

```
-1->[linenum, mac] = where()
 mac  =
!pause    !
!         !
!MyRoot2  !
 linenum  =
    0.
    6.
```

ตารางที่ 8.1 ตัวอย่างคำสั่งพื้นฐานสำหรับจัดการข้อผิดพลาด

ฟังก์ชัน	คำอธิบาย
`error`	แสดงข้อความออกทางหน้าต่างคำสั่ง แล้วหยุดการประมวลผลโปรแกรม
`errcatch`	สั่งให้การกระทำ (action) บางอย่างทำงาน เมื่อเกิดประเภทของข้อผิดพลาด (error type) ตามที่กำหนดในระหว่างที่กำลังทำการประมวลผลโปรแกรม
`iserror`	ตรวจสอบว่ามีข้อผิดพลาดตามที่กำหนดเกิดขึ้นภายในโปรแกรมหรือไม่
`errclear`	ยกเลิกการกระทำที่เป็นผลมาจากประเภทของข้อผิดพลาดที่กำหนดไว้

ผลลัพธ์ที่ได้แสดงให้ทราบว่ามีการเรียกใช้คำสั่ง `pause` ในโปรแกรม `MyRoot2` โดยโปรแกรมจะมาสิ้นสุดการประมวลผล ณ บรรทัดที่หก (คำสั่งบรรทัดที่หกยังไม่ถูกนำไปประมวลผล)

นอกจากนี้โปรแกรม SCILAB ยังได้เตรียมคำสั่งพื้นฐานอื่นๆ สำหรับจัดการข้อผิดพลาด (error handler) ตารางที่ 8.1 แสดงคำสั่งที่พบบ่อยในการเขียนโปรแกรม (ผู้สนใจสามารถศึกษารายละเอียดการใช้งานคำสั่งเหล่านี้ได้จากคำสั่ง `help`)

8.2 สรุป

ในบทนี้ได้อธิบายถึงขั้นตอนการตรวจสอบหาข้อผิดพลาดและรูปแบบการเรียกใช้งานคำสั่งต่างๆ ได้แก่ คำสั่ง `pause`, `setbpt`, และ `resume` เป็นต้น เพื่อใช้เป็นแนวทางในการตรวจสอบหาข้อผิดพลาดภายในตัวโปรแกรม ซึ่งจะช่วยทำให้ผู้พัฒนาโปรแกรมสามารถค้นหาข้อผิดพลาดที่เกิดขึ้นภายในตัวโปรแกรมได้โดยง่ายเพื่อจะได้ดำเนินการแก้ไขข้อผิดพลาดเหล่านั้นต่อไป

8.3 แบบฝึกหัดท้ายบท

8.1 จงเปรียบเทียบข้อแตกต่างระหว่างคำสั่ง `setbpt` และคำสั่ง `pause`

8.2 การอธิบายการใช้งานของคำสั่ง `dispbpt` และคำสั่ง `delbpt`

8.3 จงเปรียบเทียบข้อแตกต่างระหว่างคำสั่ง `quit` และคำสั่ง `abort`

8.4 จงเปรียบเทียบข้อแตกต่างระหว่างคำสั่ง `where` และคำสั่ง `whereami`

บทที่ 9

ตัวอย่างการประยุกต์ใช้งาน

ในบทนี้จะแสดงตัวอย่างการใช้โปรแกรม SCILAB ในการแก้ไขปัญหาทางคณิตศาสตร์ลักษณะต่างๆ เช่น การใช้งานเวกเตอร์และเมทริกซ์ตรรกะ, การดำเนินการเชิงสัญลักษณ์, การหาตัวคูณร่วมน้อยและตัวหารร่วมมาก, การแยกตัวประกอบของพหุนาม, การลดรูปพหุนาม, การหาปริพันธ์จำกัดเขต, และการแก้สมการอนุพันธ์อันดับหนึ่ง เป็นต้น เพื่อให้ผู้อ่านทราบถึงความสามารถของโปรแกรม SCILAB ในการที่จะนำมาประยุกต์ใช้ในการเรียนการสอน (ทั้งในระดับมัธยมศึกษาและระดับอุดมศึกษา) และการทำงานวิจัย โดยเฉพาะงานทางด้านวิศวกรรมและวิทยาศาสตร์

9.1 การใช้งานเวกเตอร์และเมทริกซ์ตรรกะ

ตัวดำเนินการตรรกะ (logical operator) ได้แก่ เครื่องหมาย AND “&”, เครื่องหมาย OR “|”, และเครื่องหมาย NOT “~” เป็นตัวดำเนินการที่ใช้เชื่อมความสัมพันธ์ระหว่างค่าของตัวแปรที่เกิดขึ้นโดยตัวดำเนินการสัมพันธ์ (relational operator) เช่น เครื่องหมายมากกว่า “>”, เครื่องหมายน้อยกว่า “<”, เครื่องหมายมากกว่าหรือเท่ากับ “>=”, เครื่องหมายน้อยกว่าหรือเท่ากับ “<=”, เครื่องหมายเท่ากับ “==”, และเครื่องหมายไม่เท่ากับ “<>” (หรือ “~=”) ถ้าความสัมพันธ์สอดคล้องกันผลลัพธ์ที่ได้ก็จะมีค่าเท่ากับ `T` (เป็นจริง) หรือมีค่าทางตรรกะเท่ากับค่า `1` แต่ถ้าความสัมพันธ์ไม่สอดคล้องกันผลลัพธ์ที่ได้ก็จะมีค่าเท่ากับ `F` (เป็นเท็จ) หรือมีค่าทางตรรกะเท่ากับค่า `0` โดยทั่วไปเวกเตอร์หรือเมทริกซ์ของข้อความแสดงการเปรียบเทียบ (comparison statement) สามารถที่จะถูกสร้างได้เช่นเดียวกับการสร้างเวกเตอร์หรือเมทริกซ์ของค่าสเกลาร์หรือสัญลักษณ์ทั่วไป ตามที่ได้อธิบายในบทที่ 2 ตัวอย่างเช่น

```
-->v = [1>2  1<3  2>=3  3<=3  2==3  1<>2  2~=2]
 v  =
  F T F T F T F
```

ผลลัพธ์ที่ได้คือเวกเตอร์ v ที่มีสมาชิกทั้งหมดเป็นตัวแปรบูลีน (Boolean) หรือผลลัพธ์ที่ได้จากการเปรียบเทียบ โดยทั่วไปตัวแปรลักษณะนี้จะเรียกกันว่าเวกเตอร์ตรรกะ (logical vector)

ในส่วนต่อไปนี้จะกล่าวถึงการดำเนินการทางคณิตศาสตร์กับเวกเตอร์และเมทริกซ์ตรรกะ ดังแสดงในตัวอย่างต่อไปนี้

```
-->x = [1>2  1<3  2>=3  3<=3  2==3  1<>2];
 x  =
  F T F T F T
-->2*x
 ans  =
    0.    2.    0.    2.    0.    2.
-->2*[0  1  0  1  0  1]
 ans  =
    0.    2.    0.    2.    0.    2.
```

ผลลัพธ์ที่ได้แสดงให้เห็นว่า ผลคูณของค่าสเกลาร์กับตัวแปรบูลีนที่เป็นจริงจะมีค่าเท่ากับค่าสเกลาร์นั้น ส่วนผลคูณของค่าสเกลาร์กับตัวแปรบูลีนที่เป็นเท็จจะมีค่าเท่ากับค่าศูนย์ ดังนั้นสามารถที่จะสรุปได้ว่าเมื่อนำเวกเตอร์และเมทริกซ์ตรรกะมาดำเนินการทางคณิตศาสตร์กับค่าสเกลาร์ เวกเตอร์และเมทริกซ์เหล่านี้สามารถแทนได้ด้วยเวกเตอร์และเมทริกซ์ของค่าสเกลาร์ที่มีสมาชิกเป็นค่า 0 กับค่า 1 โดยที่ค่า 0 จะใช้แทนตัวแปรบูลีนที่เป็นเท็จ และค่า 1 จะใช้แทนตัวแปรบูลีนที่เป็นจริง เพราะฉะนั้นจากตัวอย่างข้างต้นจะได้ว่า เวกเตอร์ x มีค่าเทียบเท่ากับเวกเตอร์ [0 1 0 1 0 1]

เมื่อเข้าใจถึงหลักการพื้นฐานในการดำเนินการทางคณิตศาสตร์กับเวกเตอร์และเมทริกซ์ตรรกะของโปรแกรม SCILAB แล้ว ก็จะช่วยทำให้สามารถนำเวกเตอร์และเมทริกซ์ตรรกะมาประยุกต์ใช้งานในลักษณะต่างๆ ได้อย่างถูกต้อง ตัวอย่างเช่น

```
-->v = [1>2  1<3  2>=3  3<=3  2==3  1<>2];
-->v + [1  2  3  4  5  6]        //เวกเตอร์ v เปรียบเสมือนมีค่าเท่ากับ v = [0 1 0 1 0 1]
 ans  =
    1.    3.    3.    5.    5.    7.
```

```
-->v .* [1  2  3  4  5  6]
 ans  =
    0.    2.    0.    4.    0.    6.
```

เมทริกซ์ตรรกะ คือ เมทริกซ์ที่สมาชิกทั้งหมดภายในเมทริกซ์เป็นตัวแปรบูลีน (T หรือ F) ในทำนองเดียวกันผู้ใช้สามารถดำเนินการทางคณิตศาสตร์กับเมทริกซ์ตรรกะได้เหมือนกับเวกเตอร์ตรรกะ ดังแสดงในตัวอย่างต่อไปนี้

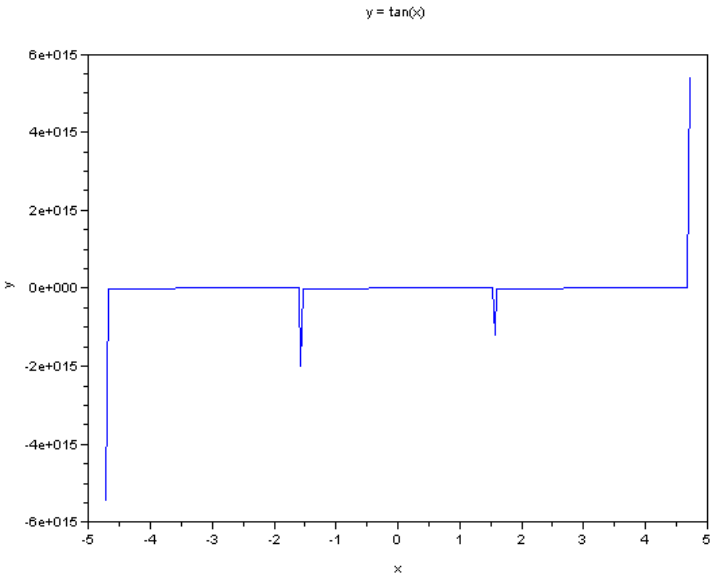
```
-->A = [1 2; 3 4]                    //เมทริกซ์ค่าสเกลาร์
 A  =
    1.    2.
    3.    4.

-->B = [1<2  2>3;  2==3  2<>3]    //เมทริกซ์ตรรกะเทียบเท่ากับเมทริกซ์ [1 0; 0 1]
 B  =
  T F
  F T

-->A + B
 ans  =
    2.    2.
    3.    5.

-->A .* B
 ans  =
    1.    0.
    0.    4.
```

9.1.1 การใช้งานเวกเตอร์และเมทริกซ์ตรรกะสำหรับวาดรูปกราฟ

เวกเตอร์และเมทริกซ์ตรรกะมีประโยชน์มากในการช่วยวาดรูปกราฟแบบต่างๆ ดังแสดงต่อไปนี้

9.1.1.1 การแก้ไขปัญหาค่าอนันต์

ในกรณีที่ข้อมูลที่จะนำมาใช้ในการวาดรูปกราฟมีค่าที่เป็นค่าอนันต์ (infinity) รวมอยู่ด้วย ถ้าวาดรูปกราฟโดยใช้ข้อมูลนี้ก็จะได้รูปกราฟที่ไม่สวยงาม ตัวอย่างเช่น จากคุณสมบัติของฟังก์ชันตรีโกณมิติจะได้ว่า $\tan(x) = \sin(x)/\cos(x) = \pm\infty$ เมื่อ $x = \pm n\pi/2$ สำหรับ $n = 1, 2, 3, \ldots$ ที่เป็น



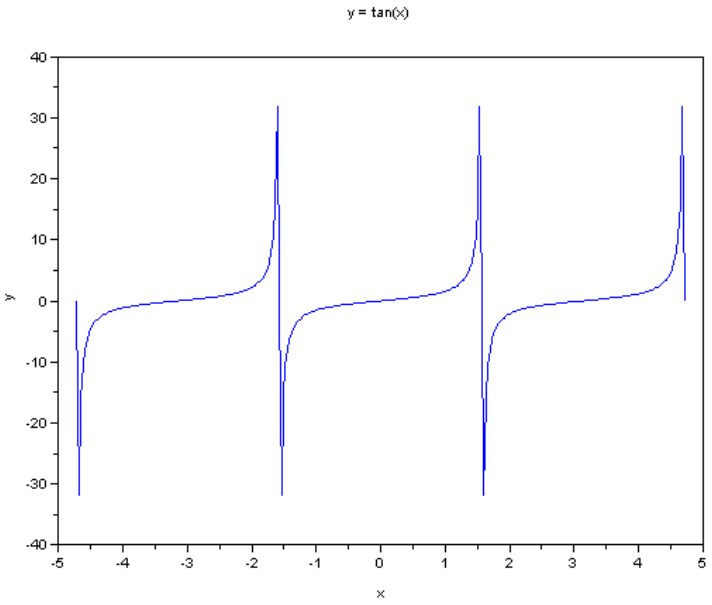
รูปที่ 9.1 รูปกราฟของฟังก์ชันตรีโกณมิติ `tan(x)`

จำนวนเต็มบวกเนื่องจาก `cos(x) = 0` ดังนั้นถ้าต้องการวาดรูปกราฟของฟังก์ชัน `tan(x)` โดยที่ `x` มีค่าตั้งแต่ $-3\pi/2$ ถึง $3\pi/2$ โดยใช้ชุดคำสั่งดังนี้

```
-->x = -3*%pi/2:%pi/100:3*%pi/2;
-->y = tan(x);
-->plot(x, y)
-->xtitle('y = tan(x)', 'x', 'y');
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 9.1 ซึ่งจะพบว่ารูปกราฟที่ได้จะมีสไปค์ (spike) เกิดขึ้นมา ณ ตำแหน่งที่ค่า $x = \pm\pi$ และ $x = \pm 3\pi/2$ เนื่องจากตัวแปร `y` มีค่าสูงมาก (ค่าเข้าใกล้ค่าอนันต์) ทำให้รูปกราฟที่ได้ไม่สามารถแสดงรายละเอียดได้ครบถ้วน วิธีการแก้ไขปัญหานี้สามารถทำได้โดยใช้คุณสมบัติของเวกเตอร์ตรรกะโดยการปรับค่าของตัวแปร `y` ดังนี้

```
-->x = -3*%pi/2:%pi/100:3*%pi/2;
-->y = tan(x);
```



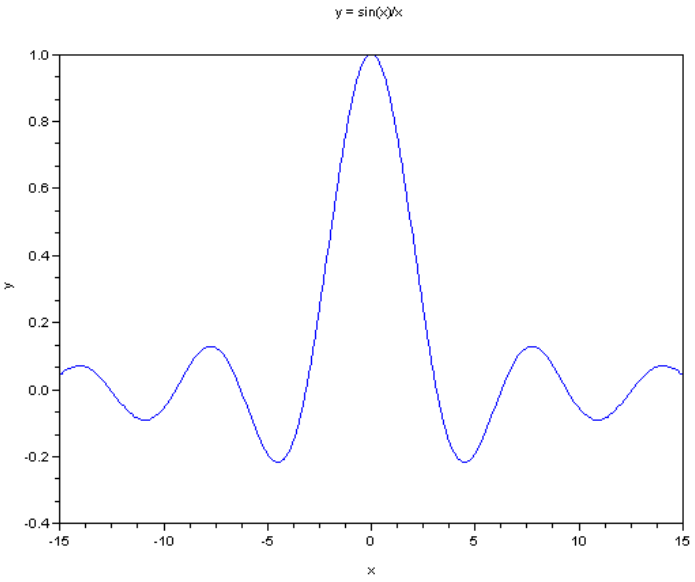
รูปที่ 9.2 รูปกราฟของฟังก์ชัน `tan(x)` หลังจากปรับค่าของตัวแปร `y`

```
-->y = y .* (abs(y) < 1e10);                    //ปรับค่าของตัวแปร y
-->plot(x, y)
-->xtitle('y = tan(x)', 'x', 'y');
```

คำสั่งบรรทัดที่สามเป็นการปรับค่าของตัวแปร `y` โดยที่ถ้าค่าสัมบูรณ์ของตัวแปร `y` มีค่าน้อยกว่าค่า 10^{10} ก็จะปรับให้ตัวแปร `y` มีค่าเท่ากับค่าศูนย์ ผลลัพธ์ที่ได้จากการใช้ชุดคำสั่งนี้แสดงในรูปที่ 9.2 ซึ่งจะแสดงรายละเอียดของกราฟได้ครบถ้วนตามที่ต้องการ

9.1.1.2 การแก้ไขปัญหาการหารด้วยค่าศูนย์

ในกรณีที่ข้อมูลที่ใช้วาดรูปกราฟมีค่าที่เป็นค่าอนันต์ (infinity) ที่เกิดจากการหารค่าจำนวนจริงด้วยค่าศูนย์ เช่น ถ้าต้องการวาดรูปกราฟของฟังก์ชันซิงก์ `y = sin(x)/x` ซึ่งเป็นสัญญาณที่พบมากในระบบการสื่อสารดิจิทัล ดังนั้นเมื่อตัวแปร `x = 0` จะทำให้ตัวแปร `y` มีค่าเป็นค่าอนันต์ ดังแสดงในตัวอย่างต่อไปนี้



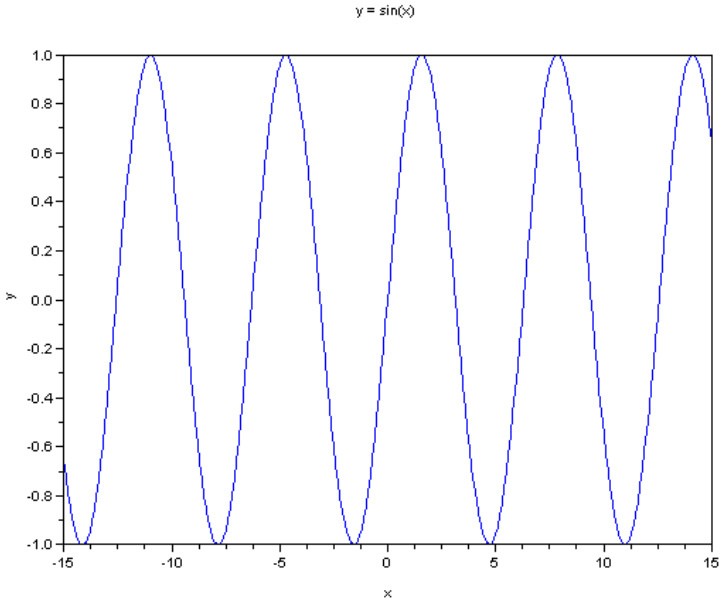
รูปที่ 9.3 รูปกราฟของฟังก์ชัน `sin(x)/x` หลังจากปรับค่าของตัวส่วนที่มีค่าเป็นค่าศูนย์

```
-->x = [-15:0.1:15];
-->y = sin(x)./x;
                --error    27
division by zero...
```

วิธีการแก้ปัญหาการหารค่าจำนวนจริงด้วยค่าศูนย์ในโปรแกรม SCILAB สามารถทำได้โดยการแทนค่าตัวส่วน (denominator) ที่มีค่าเป็นค่าศูนย์ด้วยค่า `%eps = 2.220D-16` วิธีการหาตำแหน่งของตัวส่วนที่มีค่าเป็นค่าศูนย์สามารถทำได้หลายวิธี เช่น การใช้คำสั่ง `find` หรือการใช้คุณสมบัติของเวกเตอร์ตรรกะ ดังแสดงในตัวอย่างต่อไปนี้

```
-->clf;  x = [-15:0.1:15];
-->x = x + (x == 0) * %eps;                    //ปรับค่าของตัวแปร x
-->y = sin(x) ./ x;
-->plot(x, y)
-->xtitle('y = sin(x)/x', 'x', 'y');
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 9.3



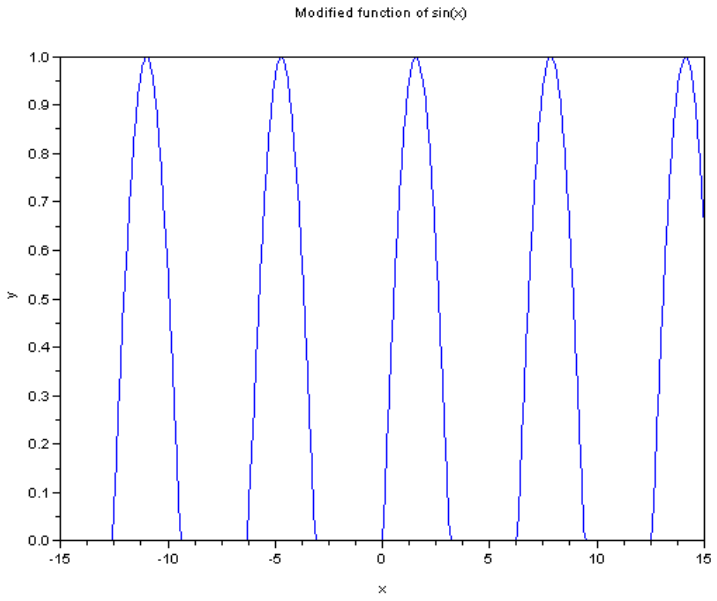
รูปที่ 9.4 รูปกราฟของฟังก์ชัน `sin(x)`

9.1.1.3 การสร้างกราฟแบบไม่ต่อเนื่อง

ในการทำงานบางครั้งมีความจำเป็นต้องเปลี่ยนแปลงรูปกราฟแบบต่อเนื่อง (continuous graph) ให้อยู่ในรูปกราฟแบบไม่ต่อเนื่อง (discontinuous graph) เพื่อนำเสนอผลการทดลองตามที่กำหนด ตัวอย่างเช่น รูปกราฟของฟังก์ชันตรีโกณมิติ `sin(x)` เป็นกราฟแบบต่อเนื่องซึ่งมีค่าทั้งด้านบวกและด้านลบตามรูปที่ 9.4

```
-->clf;  x = [-15:0.1:15];
-->y = sin(x);
-->plot(x, y)
-->xtitle('y = sin(x)', 'x', 'y');
```

คุณสมบัติของเวกเตอร์ตรรกะสามารถที่จะถูกนำมาใช้ในการเปลี่ยนรูปกราฟนี้ให้เป็นกราฟแบบไม่ต่อเนื่อง เช่น เปลี่ยนให้เป็นรูปกราฟที่แสดงผลเฉพาะค่าบวกของสัญญาณ ดังแสดงในตัวอย่างต่อไปนี้



รูปที่ 9.5 รูปกราฟของฟังก์ชัน `sin(x)` ที่แสดงผลเฉพาะค่าบวก

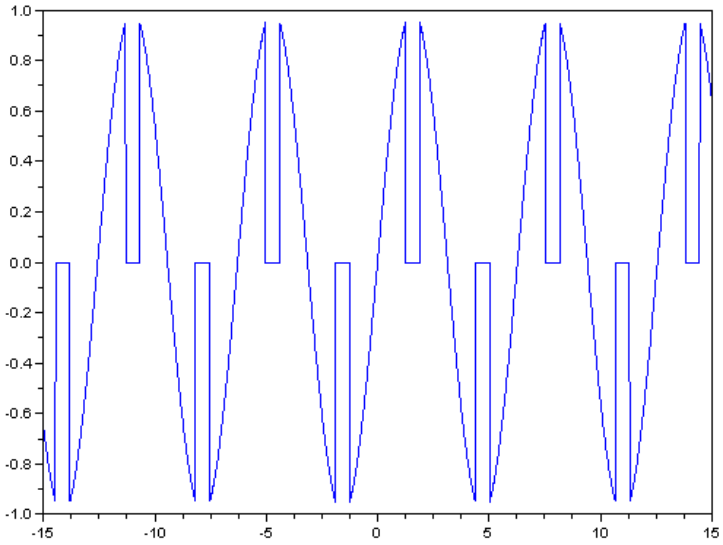
```
-->clf;  x = [-15:0.1:15];
-->y = sin(x);
-->y = y .* (y >= 0);
-->plot(x, y)
-->xtitle('Modified function of sin(x)', 'x', 'y');
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 9.5 ซึ่งเป็นรูปสัญญาณที่พบมากในงานทางด้านวิศวกรรมไฟฟ้า เช่น รูปสัญญาณของวงจรเรียงกระแสตรงแบบครึ่งคลื่น (half-wave rectifier) เป็นต้น

9.1.1.4 การขริบข้อมูลบางส่วนของกราฟ

ในทำนองเดียวกันกับหัวข้อที่ 9.1.1.3 ผู้ใช้สามารถที่จะขริบหรือเล็ม (clip) ข้อมูลที่แสดงผลบางส่วนของรูปกราฟได้โดยใช้คุณสมบัติของเวกเตอร์ตรรกะ ดังแสดงในตัวอย่างต่อไปนี้

```
-->clf;  x = [-15:0.01:15];
-->y = sin(x);
```



รูปที่ 9.6 ตัวอย่างรูปกราฟของฟังก์ชัน `sin(x)` ที่ตัดข้อมูลที่แสดงผลบางส่วนแบบที่หนึ่ง

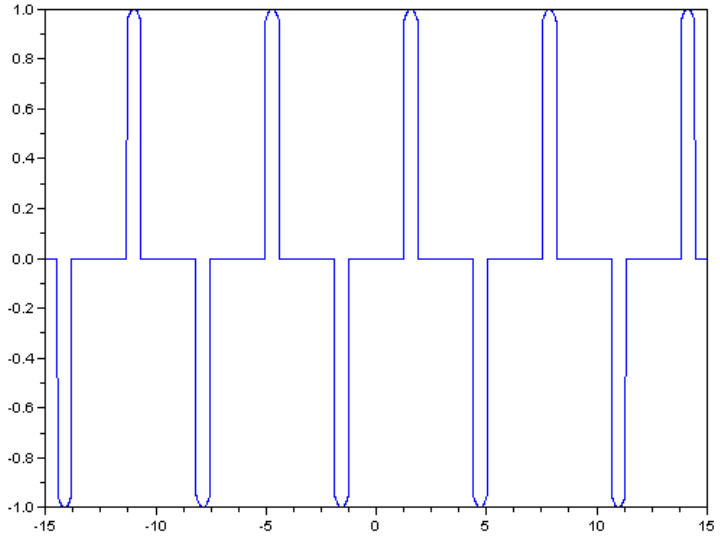
```
-->x = [-15:0.01:15];
-->y = sin(x);
-->y = y .* (abs(y) <= 0.95);
-->plot(x, y)
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 9.6 หรืออีกตัวอย่างหนึ่งคือ

```
-->clf;  x = [-15:0.01:15];
-->y = sin(x);
-->y = y .* (y<= -0.95 | y>= 0.95);
-->plot(x,y)
```

ผลลัพธ์ที่ได้แสดงในรูปที่ 9.7

จากตัวอย่างต่างๆ ที่แสดงข้างต้นแสดงให้เห็นว่าเวกเตอร์และเมทริกซ์ตรรกะสามารถที่จะนำมาประยุกต์ใช้ในการวาดรูปกราฟแบบต่างๆ ได้ตรงตามความต้องการ



รูปที่ 9.7 ตัวอย่างรูปกราฟของฟังก์ชัน `sin(x)` ที่ตัดข้อมูลที่แสดงผลบางส่วนแบบที่สอง

9.2 การดำเนินการเชิงสัญลักษณ์

โปรแกรม SCILAB รองรับการดำเนินการเชิงสัญลักษณ์ (symbolic operation) ได้ใกล้เคียงกับโปรแกรมอื่นๆ เช่น Maple, MuPad, และ Mathematica เป็นต้น ทำให้สามารถพัฒนาโปรแกรมได้หลากหลายรูปแบบตามที่ต้องการ ในส่วนนี้จะกล่าวถึงตัวอย่างการประยุกต์ใช้งานที่เกี่ยวข้องกับการดำเนินการเชิงสัญลักษณ์ดังต่อไปนี้

9.2.1 ฟังก์ชัน `addf`, `subf`, `mulf`, `ldivf`, และ `rdivf`

ฟังก์ชันเหล่านี้จะทำหน้าที่ในการดำเนินการเชิงสัญลักษณ์ ได้แก่ การบวก “`addf`”, การลบ “`subf`”, การคูณ “`mulf`”, การหารซ้าย “`ldivf`”, และการหารขวา “`rdivf`” ระหว่างสายอักขระ (character string) โดยผลลัพธ์ที่ได้จะเป็นสายอักขระใหม่ ตัวอย่างเช่น

```
-->addf('x', '1')                          //การบวกเชิงสัญลักษณ์
 ans  =
 x+1
```

```
-->addf('x+2', 'y-3')                    //การบวกเชิงสัญลักษณ์
 ans  =
 x+y-1

-->subf('x+2', 'y-3')                    //การลบเชิงสัญลักษณ์
 ans  =
 x-y+5

-->mulf('x+2', 'y-3')                    //การคูณเชิงสัญลักษณ์
 ans  =
 (x+2)*(y-3)

-->ldivf('x+2', 'y-3')                   //การหารซ้ายเชิงสัญลักษณ์
 ans  =
 (x+2)\(y-3)

-->rdivf('x+2', 'y-3')                   //การหารขวาเชิงสัญลักษณ์
 ans  =
 (x+2)/(y-3)
```

9.2.2 ฟังก์ชัน cmb_lin

เป็นฟังก์ชันที่ทำหน้าที่ในการสร้างสมการเชิงเส้นเชิงสัญลักษณ์ (symbolic linear equation) โดยมีรูปแบบการเรียกใช้งานคือ

```
z = cmb_lin(a, x, b, y)
```

ผลลัพธ์ที่ได้คือ z = a*x – b*y เมื่อพารามิเตอร์ z, a, x, b, และ y เป็นสายอักขระ ตัวอย่างเช่น

```
-->z = cmb_lin('3', 'x', '2', 'y')
 z  =
 3*x-2*y

-->z = cmb_lin('A', 'x^2+x+1', '2*B', '2*y-1')
 z  =
 A*(x^2+x+1)-2*B*(2*y-1)
```

9.2.3 ฟังก์ชัน eval และ evstr

เป็นฟังก์ชันที่ทำหน้าที่ในการคำนวณหาค่าผลลัพธ์ของสมการเชิงเส้นเชิงสัญลักษณ์ที่สร้างขึ้น เมื่อกำหนดค่าจำนวนจริงให้กับสัญลักษณ์ที่ใช้ในสมการเชิงเส้น ตัวอย่างเช่น

```
-->y = cmb_lin(mulf('2', 'x'), 'x^2 - 1', addf('3','2*x'), 'x^3')
 y  =
 2*x*(x^2-1)-(2*x+3)*x^3                    //ตัวแปร y เป็นสายอักขระ

-->x = 3;                                   //กำหนดให้ตัวแปร x มีค่าเท่ากับค่า 3

-->eval(y)
 ans  =
  - 195.        //ผลลัพธ์ที่ได้คือ 2×3×(3^2−1)−(2×3+3)×3^3 = 6×8−9×27 = 48−243 = −195

-->x = poly(0, 'x');                        //กำหนดให้ตัวแปร x เป็นตัวแปรพหุนาม

-->y
 y  =
 2*x*(x^2-1)-(2*x+3)*x^3

-->eval(y)
 ans  =
          3     4
  - 2x - x - 2x                             //ตัวแปร y ถูกเปลี่ยนเป็นสมการพหุนาม

-->evstr(y)                                 //มีผลลัพธ์เท่ากับการใช้คำสั่ง eval
 ans  =
          3     4
  - 2x - x - 2x

-->x = 4;

-->eval(y)                                  //คำนวณหาค่า y เมื่อกำหนดค่าตัวแปร x = 4
 ans  =
  - 584.

-->evstr(y)                                 //มีผลลัพธ์เท่ากับการใช้คำสั่ง eval
 ans  =
  - 584.
```

9.2.4 ฟังก์ชัน trianfml

เป็นฟังก์ชันที่ทำหน้าที่ในการทำ symbolic triangularization ของเมทริกซ์ นั่นคือการทำให้สมาชิกของเมทริกซ์ที่อยู่ใต้เส้นทแยงมุมหลัก (main diagonal) มีค่าเป็นค่าศูนย์ หรืออาจจะกล่าวได้ว่าเป็นการทำให้เมทริกซ์เชิงสัญลักษณ์เป็นเมทริกซ์สามเหลี่ยมด้านบน (upper triangular matrix) โดยวิธีทางคณิตศาสตร์ที่เรียกว่าการดำเนินการตามแถวขั้นมูลฐาน (elementary row operation) ตัวอย่างเช่น (ดูรายละเอียดเพิ่มเติมในหัวข้อที่ 2.3)

```
-->A = ['a11', 'a12';  'a21', 'a22']
 A  =
!a11  a12  !
!          !
!a21  a22  !

-->B = trianfml(A)
 B  =
!a21  a22              !
!                      !
!0    a21*a12-a11*a22  !

-->a11=1; a12=2; a21=3; a22=4;

-->evstr(A)
 ans  =
    1.    2.
    3.    4.

-->evstr(B)
 ans  =
    3.    4.
    0.    2.
```

9.2.5 ฟังก์ชัน solve

เป็นฟังก์ชันที่ทำหน้าที่ในการหาผลเฉลยเชิงสัญลักษณ์ (หาค่าของตัวแปร x) ของสมการที่อยู่ในรูปของเมทริกซ์ Ax = b โดยที่ b เป็นเวกเตอร์เชิงสัญลักษณ์ และ A เป็นเมทริกซ์สามเหลี่ยมด้านบนเชิงสัญลักษณ์ ตัวอย่างเช่น

```
-->A = ['a11', 'a12';  '0', 'a22']
 A  =
!a11  a12  !
!          !
!0    a22  !

-->b = ['b1';  'b2']
 b  =
!b1  !
!    !
!b2  !

-->x = solve(A,b)
 x  =
!a11\(b1-a12*(a22\b2))  !
!                       !
!a22\b2                 !

-->a11=1; a12=1; a22=2; b1=2; b2=5;

-->evstr(x)
 ans  =
  - 0.5
    2.5
```

จากตัวอย่างที่แสดงข้างต้นพบว่าโปรแกรม SCILAB สามารถทำงานกับข้อมูลเชิงสัญลักษณ์ได้อย่างมีประสิทธิภาพ ซึ่งจะช่วยทำให้นักพัฒนาโปรแกรมสามารถที่จะพัฒนาโปรแกรมแบบใหม่ๆ ขึ้นมาใช้งานได้อย่างมีประสิทธิภาพ

9.3 ตัวคูณร่วมน้อยและตัวหารร่วมมาก

ในส่วนนี้จะแสดงการใช้งานคำสั่งของโปรแกรม SCILAB สำหรับการหาค่าตัวคูณร่วมน้อย (least common multiple) และตัวหารร่วมมาก (great common divisor) ซึ่งจะเป็นประโยชน์มากในการแก้ไขปัญหาทางคณิตศาสตร์

9.3.1 ตัวคูณร่วมน้อย

ตัวคูณร่วมน้อย (ค.ร.น.) คือเลขจำนวนเต็มบวกที่มีค่าน้อยที่สุด ซึ่งเมื่อนำเลขจำนวนเต็มบวกอื่นๆ ที่กำหนดให้ทุกจำนวนมาหารตัวคูณร่วมน้อยนี้แล้วจะได้ผลลัพธ์ลงตัวพอดี (หรือสามารถหารลงตัวได้ทุกจำนวน) การหาตัวคูณร่วมน้อยในโปรแกรม SCILAB สามารถทำได้โดยใช้คำสั่ง `lcm` ซึ่งมีรูปแบบการใช้งานคือ

```
pp = lcm(p)
```

เมื่อ `p` คือเวกเตอร์ของเลขจำนวนเต็มบวก (positive integer) หรือเวกเตอร์ของพหุนาม (polynomial) และ `pp` คือตัวคูณร่วมน้อยของสมาชิกทั้งหมดที่อยู่ในเวกเตอร์ `p`

ตัวอย่างที่ 1 จงหาตัวคูณร่วมน้อยของ 12, 15, และ 30

วิธีทำ เนื่องจาก $12 = 2\times2\times3$, $15 = 3\times5$, และ $30 = 2\times3\times5$ ดังนั้นตัวคูณร่วมน้อยของ 12, 15, และ 30 คือ $2\times2\times3\times5 = 60$ ผู้ใช้สามารถตรวจคำตอบได้จากการใช้ชุดคำสั่งต่อไปนี้

```
-->p = int16([12 15 30])          //ทำให้เป็นเลขจำนวนเต็มขนาด 16 บิต
p  =
  12  15  30
-->pp = lcm(p)
 pp  =
  60                               //ไม่มีจุดทศนิยมตามหลังตัวเลขแสดงว่าเป็นเลขจำนวนเต็ม
```

ซึ่งให้ผลลัพธ์ตรงตามที่ต้องการ

หมายเหตุ ถ้าสมาชิกในเวกเตอร์ `p` ที่ใช้ในคำสั่ง `lcm(p)` เป็นเลขจำนวนจริง ก็จะทำให้ได้ผลลัพธ์ที่ผิดไปจากที่คาดหวังไว้ เช่น

```
-->p = [12 15 30]
 p  =
    12.    15.    30.             //มีจุดทศนิยมตามหลังตัวเลขแสดงว่าเป็นเลขจำนวนจริง
```

```
-->pp = lcm(p)
 pp  =

    5400.                                    //ผลลัพธ์ที่ได้คือ 5400 = 12×15×30
```

นอกจากนี้คำสั่ง lcm ยังสามารถใช้หาตัวคูณร่วมน้อยของพหุนามได้ ดังแสดงในตัวอย่างต่อไปนี้

ตัวอย่างที่ 2 จงหาตัวคูณร่วมน้อยของพหุนาม x, x^2+x, และ x^2-x

วิธีทำ เนื่องจาก $x^2+x=x(x+1)$ และ $x^2-x=x(x-1)$ เพราะฉะนั้นตัวคูณร่วมน้อยของ x, x^2+x, และ x^2-x มีค่าเท่ากับ $x(x+1)(x-1)=x^3-x$ ซึ่งผู้ใช้สามารถตรวจคำตอบได้จากการใช้ชุดคำสั่งดังนี้

```
-->x = poly(0,'x');                          //กำหนดให้ x เป็นตัวแปรพหุนาม

-->p = [x,  x^2+x,  x^2-x]
 p  =
                2          2
    x     x + x    - x + x

-->pp = lcm(p)
 pp  =
          3
  - x + x
```

ซึ่งให้ผลลัพธ์เท่ากัน

9.3.2 ตัวหารร่วมมาก

ตัวหารร่วมมาก (ห.ร.ม.) คือเลขจำนวนเต็มบวกที่มีค่ามากที่สุด ซึ่งเมื่อนำมาหารเลขจำนวนเต็มบวกอื่นๆ ที่กำหนดให้ทุกจำนวนแล้วปรากฏว่าหารลงตัวได้ทุกจำนวน การหาตัวหารร่วมมากในโปรแกรม SCILAB สามารถทำได้โดยใช้คำสั่ง gcd ซึ่งมีรูปแบบการใช้งานคือ

```
qq = gcd(q)
```

เมื่อ q คือเวกเตอร์ของเลขจำนวนเต็มบวกหรือเวกเตอร์ของพหุนาม และ qq คือตัวหารร่วมมากของสมาชิกทั้งหมดที่อยู่ในเวกเตอร์ q

---

ตัวอย่างที่ 3 จงหาตัวหารร่วมมากของ 18 และ 84

วิธีทำ เนื่องจาก $18 = 2\times3\times3$ และ $84 = 2\times2\times3\times7$ ดังนั้นตัวหารร่วมมากของ 18 และ 84 คือ $2\times3 = 6$ ผู้ใช้สามารถตรวจคำตอบได้จากการใช้ชุดคำสั่งต่อไปนี้

```
-->q = int32([18, 84])              //ทำให้เป็นเลขจำนวนเต็มขนาด 32 บิต
q  =
  18   84
-->qq = gcd(q)
 qq  =
  6                                 //ไม่มีจุดทศนิยมตามหลังตัวเลขแสดงว่าเป็นเลขจำนวนเต็ม
```

ซึ่งให้ผลลัพธ์ตรงตามที่ต้องการ

หมายเหตุ ในทำนองเดียวกันถ้าสมาชิกในเวกเตอร์ q ที่ใช้ในคำสั่ง gcd(q) เป็นเลขจำนวนจริงก็จะทำให้ได้ผลลัพธ์ที่ผิดไปจากที่คาดหวังไว้ เช่น

```
-->q = [18, 84]
 q  =
    18.    84.                      //มีจุดทศนิยมตามหลังตัวเลขแสดงว่าเป็นเลขจำนวนจริง
-->qq = gcd(q)
 qq  =
    1                               //เลขจำนวนเต็มค่า 1 เป็นตัวหารร่วมมากของเลขทุกจำนวน
```

นอกจากนี้คำสั่ง gcd ยังสามารถใช้หาตัวหารร่วมมากของพหุนามได้เช่นกัน ดังแสดงในตัวอย่างต่อไปนี้

ตัวอย่างที่ 4 จงหาตัวคูณร่วมน้อยและตัวหารร่วมมากของพหุนาม $x^2-5x-14$ และ $x^2-12x+35$

วิธีทำ เนื่องจาก $x^2-5x-14=(x-7)(x+2)$ และ $x^2-12x+35=(x-7)(x-5)$ ดังนั้นจะได้ว่าตัวคูณร่วมน้อยมีค่าเท่ากับ $(x-7)(x+2)(x-5)=x^3-10x^2+11x+70$ และตัวหารร่วมมากมีค่าเท่ากับ $(x-7)$ ซึ่งผู้ใช้สามารถตรวจคำตอบได้จากการใช้ชุดคำสั่งต่อไปนี้

```
-->x = poly(0, 'x');                          //กำหนดให้ x เป็นตัวแปรพหุนาม

-->p = [x^2-5*x-14, x^2-12*x+35]
 p  =
                2                 2
  - 14 - 5x + x     35 - 12x + x

-->lcm(p)
 ans  =
                   2   3
    70 + 11x - 10x + x

-->gcd(p)
 ans  =
  - 7 + x
```

ซึ่งให้ผลลัพธ์ตรงตามที่ต้องการ

9.4 พหุนาม

ในส่วนนี้จะยกตัวอย่างการใช้งานคำสั่งต่างๆ ของโปรแกรม SCILAB สำหรับการจัดการกับพหุนาม เช่น การแยกตัวประกอบของพหุนามให้อยู่ในรูปการคูณของพหุนามที่มีดีกรี[29] ต่ำกว่า และการลดรูปพหุนาม เป็นต้น

[29] โปรแกรม SCILAB สามารถคำนวณหาค่าดีกรีของพหุนาม (degree of polynomial) ได้จากคำสั่ง `degree`

9.4.1 การแยกตัวประกอบของพหุนาม

การแยกตัวประกอบของพหุนาม (polynomial factorization) คือ การเขียนพหุนามให้อยู่ในรูปการคูณของพหุนามที่มีดีกรีต่ำกว่า ตัวอย่างเช่น พหุนาม x^2-3x+2 สามารถที่จะแยกตัวประกอบได้เป็น $(x-1)(x-2)$ นั่นคือ $x^2-3x+2=(x-1)(x-2)$ ในโปรแกรม SCILAB การแยกตัวประกอบของพหุนามสามารถทำได้โดยใช้คำสั่ง `factors` ซึ่งมีรูปแบบการใช้งานดังนี้

```
[F, g] = factors(y)
```

โดยที่ `y` คือสมการพหุนามที่เป็นฟังก์ชันของตัวแปรเพียงตัวแปรเดียว, `g` คือเลขจำนวนจริง, และ `F` คือเวกเตอร์พหุนาม (ที่มีดีกรีหนึ่งหรือสองเท่านั้น) ถ้ากำหนดให้ `F = [f1 f2 f3 ... fN]` เมื่อ `fi` คือพหุนาม ดังนั้นผลลัพธ์ที่ได้จากการใช้คำสั่ง `factors` คือ

$$y = g \times f_1 \times f_2 \times f_3 \times \ldots \times f_N$$

ตัวอย่างเช่น

```
-->x = poly(0, 'x');
-->y = x^2 - 3*x + 2
 y  =
                2
    2 - 3x + x

-->[F, g] = factors(y)              //ผลลัพธ์ที่ได้คือ x^2 - 3x + 2 = 1×(x-1)×(x-2)
 g  =
    1.
 F  =
       F(1)
  - 1 + x
       F(2)
  - 2 + x

-->F = polfact(y)                   //มีผลลัพธ์เทียบเท่ากับการใช้คำสั่ง factors
```

```
 F  =
    1    - 2 + x    - 1 + x
```

จะเห็นได้ว่าคำสั่ง polfact สามารถทำหน้าที่ในการแยกตัวประกอบของพหุนามได้เช่นเดียวกับคำสั่ง factors แต่มีรูปแบบการเรียกใช้งานต่างกันเล็กน้อย นั่นคือ

```
P = polfact(y)
```

เมื่อ P = [p_0 p_1 p_2 ... p_N] คือเวกเตอร์พหุนาม (ที่มีดีกรีหนึ่งหรือสองเท่านั้น) และ p_0 เป็นค่าคงตัว (constant) โดยผลลัพธ์ที่ได้จากการใช้คำสั่ง polfact คือ

$$y = p_0 \times p_1 \times p_2 \times \ldots \times p_N$$

ตัวอย่างเช่น

```
-->x = poly(0, 'x');
-->y = 3*x^2 - 9*x + 6;
-->[F, g] = factors(y)
 g  =
    3.                                  //ผลลัพธ์ที่ได้คือ 3x^2 - 9x + 6 = 3×(x-1)×(x-2)
 F  =
       F(1)
  - 1 + x
       F(2)
  - 2 + x
-->P = polfact(y)                       //มีผลลัพธ์เทียบเท่ากับการใช้คำสั่ง factors
 P  =
    3    - 2 + x    - 1 + x
```

ในกรณีที่ต้องการแยกตัวประกอบของพหุนามที่อยู่ในรูปของเศษส่วน (rational polynomial) ก็สามารถทำได้โดยใช้รูปแบบการเรียกใช้งานคำสั่ง factors ดังนี้

```
[N, D, g] = factors(Q)
```

เมื่อ Q คือสมการพหุนามที่อยู่ในรูปของเศษส่วน, g คือเลขจำนวนจริง, และ N คือเวกเตอร์พหุนามของตัวเศษ (numerator), และ D คือเวกเตอร์พหุนามของตัวส่วน (denominator) ในทำนองเดียวกัน ถ้าให้ $N = [n_1\ n_2\ n_3 \ldots n_k]$ และ $D = [d_1\ d_2\ d_3 \ldots d_m]$ เมื่อ n_i และ d_j คือพหุนาม (ที่มีดีกรีหนึ่งหรือสองเท่านั้น) ดังนั้นผลลัพธ์ที่ได้จากการใช้คำสั่ง factors ในกรณีนี้คือ

$$Q = g \times \left(\frac{n_1 \times n_2 \times \ldots \times n_k}{d_1 \times d_2 \times \ldots \times d_m} \right)$$

ตัวอย่างเช่น

```
-->x = poly(0, 'x');
-->Q = (3 *x^2 - 9*x + 6)/(x^2 + 3*x + 2)
 Q  =
                2
    6 - 9x + 3x
    -----------
                2
    2 + 3x + x
-->[N, D, g] = factors(Q)
 g  =
    3.
 D  =
       D(1)
    1 + x
       D(2)
    2 + x
 N  =
       N(1)
  - 1 + x
       N(2)
  - 2 + x
```

//ผลลัพธ์ที่ได้คือ $\frac{3x^2-9x+6}{x^2+3x+2} = 3 \times \frac{(x-1)\times(x-2)}{(x+1)\times(x+2)}$

9.4.2 การลดรูปพหุนาม

โปรแกรม SCILAB ยังทำการลดรูปของพหุนามที่เป็นเศษส่วนให้อยู่ในรูปแบบที่ง่าย (simple form) โดยใช้คำสั่ง `simp` ซึ่งมีรูปแบบการใช้งานคือ

```
[n, d] = simp(Num, Den)
```

เมื่อ `Num` และ `Den` คือตัวเศษและตัวส่วนของพหุนามที่ต้องการจะลดรูป ในขณะที่ `n` และ `d` คือตัวเศษและตัวส่วนของพหุนามที่ลดรูปเสร็จแล้ว ตัวอย่างเช่น

```
-->x = poly(0, 'x');
-->[n, d] = simp((9*x^3 - 21*x^2 + 16*x - 4), (x-1))
 d  =
    1
 n  =                    //ผลลัพธ์ที่ได้คือ
                    2
    4 - 12x + 9x
-->[n, d] = simp((9*x^2 - 12*x + 4), (3*x-2)*(x+1))
d  =
    1 + x
 n  =
  - 2 + 3x
```

$$\frac{9x^3-21x^2+16x-4}{x-1} = \frac{(x-1)\times(3x-2)^2}{(x-1)} = \frac{9x^2-12x+4}{1}$$

9.4.3 คำสั่งที่น่าสนใจ

โปรแกรม SCILAB ยังได้เตรียมคำสั่งต่างๆ ที่น่าสนใจในการทำงานกับพหุนาม เช่น `cmndred`, `pdiv`, `coffg`, และ `coeff` เป็นต้น ซึ่งมีรายละเอียดดังต่อไปนี้

9.4.3.1 คำสั่ง `cmndred`

เป็นคำสั่งที่ใช้ในการจัดรูปผลบวกของพหุนามที่อยู่ในรูปของเศษส่วน ให้เป็นผลบวกของพหุนามที่มีตัวส่วนร่วมเหมือนกันทุกพจน์ โดยมีรูปแบบการเรียกใช้งานดังนี้

```
[n, d] = cmndred(N, D)
```

เมื่อ N และ D คือเวกเตอร์พหุนามของตัวเศษและตัวส่วนที่ต้องการจะหาตัวส่วนร่วม ในขณะที่ n และ d คือเวกเตอร์พหุนามของตัวเศษและตัวส่วนที่มีตัวส่วนร่วมเหมือนกันทุกพจน์ ตัวอย่างเช่น พิจารณาผลบวกของพหุนาม

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

ซึ่งสามารถที่จะจัดให้อยู่ในรูปผลบวกของพหุนามที่มีตัวส่วนร่วมเหมือนกันทุกพจน์ได้ดังนี้

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

$$=\frac{x^2+2x-3}{x^2-9}+\frac{x^2-x-6}{x^2-9}$$

ในโปรแกรม SCILAB สามารถหาผลเฉลยนี้ได้โดยใช้ชุดคำสั่งต่อไปนี้

```
-->x = poly(0, 'x');

-->N = [x-1, x+2]
 N  =
  - 1 + x     2 + x

-->D = [x-3, x+3]
 D  =
  - 3 + x     3 + x

-->[n, d] = cmndred(N, D)
 d  =
          2
  - 9 + x

 n  =
               2              2
  - 3 + 2x + x    - 6 - x + x
```

```
-->[n, d] = cmndred([x-1, x+2], [x-3 x-3])
 d  =
  - 3 + x
 n  =
  - 1 + x     2 + x

-->[n, d] = cmndred([x-1 x+2 x-3], [x-3 x+2 x-3])
 d  =
              2
  - 6 - x + x
 n  =
              2             2             2
  - 2 + x + x   - 6 - x + x   - 6 - x + x
```

9.4.3.2 คำสั่ง pdiv

เป็นคำสั่งที่ใช้ในการคำนวณหาผลลัพธ์ที่ได้จากการหารพหุนาม p1 ด้วยพหุนาม p2 ซึ่งมีรูปแบบการเรียกใช้งานดังนี้

```
[r, q] = pdiv(p1, p2)
```

เมื่อ r คือผลลัพธ์ที่ได้จากการหาร และ q คือเศษที่ได้จากการหาร ตัวอย่างเช่น

```
-->x = poly(0, 'x');
-->p1 = (x^2+1)*(x-1)
 p1  =
              2   3
  - 1 + x - x + x

-->p2 = x-1
 p2  =
  - 1 + x

-->[r, q] = pdiv(p1, p2)
 q  =
         2
    1 + x
```

```
 r  =
    0.
-->[r, q] = pdiv((x^3+3*x^2-3*x+5), (x+2))
 q  =
              2
  - 5 + x + x
 r  =
    15.
```

9.4.3.3 คำสั่ง coffg

เป็นคำสั่งที่ใช้ในการหาค่าอินเวอร์สการคูณของเมทริกซ์พหุนาม ซึ่งมีรูปแบบการเรียกใช้งานดังนี้

```
[MI, d] = coffg(M)
```

เมื่อ M คือเมทริกซ์พหุนามที่ต้องการหาค่าอินเวอร์สการคูณ (ต้องเป็นเมทริกซ์จัตุรัสเท่านั้น), MI คืออินเวอร์สการคูณของเมทริกซ์ M, และ d คือตัวหารร่วมของเมทริกซ์ MI เพื่อที่ว่า

$$M = \frac{MI}{d}$$

ตัวอย่างเช่น

```
-->x = poly(0, 'x');
-->M = [x, x^2+1;  x, x^2-1]
 M  =
                2
    x     1 + x
                2
    x   - 1 + x
-->[MI, d] = coffg(M)
 d  =
  - 2x
 MI  =
         2          2
  - 1 + x    - 1 - x
  - x          x
```

```
-->I = M*MI/d                          //ตรวจคำตอบ M = MI/d
 I  =                                  //ผลลัพธ์ที่ได้คือ เมทริกซ์เอกลักษณ์ (identity matrix)
  - 2      0
    -      -
  - 2      1

    0    - 2
    -      -
    1    - 2
```

9.4.3.4 คำสั่ง coeff

เป็นคำสั่งที่ใช้ในการเรียกดูค่าสัมประสิทธิ์ (coefficient) ของพหุนาม ซึ่งมีรูปแบบการใช้งานคือ

```
C = coeff(M, [deg])
```

เมื่อ C คือเมทริกซ์ที่มีสมาชิกเป็นค่าสัมประสิทธิ์ของพหุนาม (เรียงจากดีกรีน้อยไปมาก), M คือเมทริกซ์พหุนามที่ต้องการหาค่าสัมประสิทธิ์ และ deg เป็นตัวเลือกที่ใช้ในการระบุดีกรีของพหุนามที่ต้องการจะหาค่าสัมประสิทธิ์ ถ้าไม่ได้กำหนดค่าพารามิเตอร์ deg คำสั่งนี้จะแสดงค่าสัมประสิทธิ์ทั้งหมดของพารามิเตอร์ M ตัวอย่างเช่น

```
-->x = poly(0, 'x');
-->p = 8*x^4 - 6*x^3 + 4*x^2 - 2*x + 9
 p  =
                2      3     4
    9 - 2x + 4x  - 6x  + 8x

-->c = coeff(p)                        //ค่าสัมประสิทธิ์ของพุนามจะเรียงจากดีกรีน้อยไปหาดีกรีมาก
 c  =
    9.   - 2.    4.   - 6.    8.

-->c = coeff(p, [4 2 0])               //แสดงค่าสัมประสิทธิ์ของพุนามเฉพาะดีกรี 4, 2, และ 0
 c  =
    8.    4.    9.
```

```
-->M = [6*x^3+4*x^2+2*x+1, 3*x^2-5;  -x^2+3, 3*x^3+2*x^2+x]
 M  =
                 2     3          2
    1 + 2x + 4x + 6x   - 5 + 3x

          2                      2    3
    3 - x                 x + 2x + 3x

-->C = coeff(M)
 C  =
    1.   - 5.    2.    0.    4.    3.    6.    0.
    3.    0.    0.    1.  - 1.    2.    0.    3.
```

จากผลลัพธ์จะเห็นได้ว่าในกรณีที่เป็นการหาค่าสัมประสิทธิ์ของเมทริกซ์พหุนาม คำสั่งนี้จะนำเอาพหุนามในแต่ละแถวมาหาค่าสัมประสิทธิ์ โดยจะแสดงค่าสัมประสิทธิ์ของพหุนามที่มีดีกรีน้อยที่สุดในแต่ละแนวตั้ง (เช่น ในที่นี้มีสองแนวตั้งก็จะได้เป็น แนวตั้งที่หนึ่ง แนวตั้งที่สอง, แนวตั้งที่หนึ่ง แนวตั้งที่สอง, สลับกันไป) เรียงไปจนถึงค่าสัมประสิทธิ์ของพหุนามที่มีดีกรีมาก ตามที่แสดงในตัวอย่างข้างต้น (สังเกตส่วนที่แรเงาจะได้เข้าใจมากขึ้น)

นอกจากนี้โปรแกรม SCILAB ยังได้เตรียมคำสั่งอื่นๆ ที่เกี่ยวข้องกับการจัดการพหุนาม ซึ่งผู้สนใจสามารถศึกษารายละเอียดเพิ่มเติมของคำสั่งต่างๆ ได้จากคำสั่ง `help polynomial`

9.5 การหาปริพันธ์จำกัดเขต

ปริพันธ์ (integral) คือ ฟังก์ชันที่ใช้หาพื้นที่, มวล, ปริมาตร, หรือผลรวมต่างๆ โปรแกรม SCILAB มีคำสั่งที่เกี่ยงข้องกับการหาปริพันธ์หลายคำสั่ง แต่ในที่นี้จะอธิบายเฉพาะการหาปริพันธ์จำกัดเขต[30] (definite integral) ซึ่งมีรูปแบบการใช้งานดังนี้

```
y = integrate('fx', 'x', x0, x1)
```

เมื่อ `y` คือผลลัพธ์ที่ได้จากการหาปริพันธ์, `fx` คือฟังก์ชันที่ต้องการหาปริพันธ์, `x` คือตัวแปรของการหาปริพันธ์, `x0` และ `x1` คือค่าเริ่มต้นและค่าสุดท้ายของตัวแปร `x` ที่ใช้หาปริพันธ์

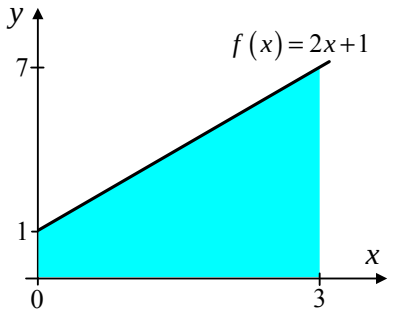
[30] ปริพันธ์จำกัดเขตจะมีช่วงของการหาปริพันธ์กำหนดมาให้ให้

นอกจากนี้ในกรณีที่ต้องการหาปริพันธ์จำกัดเขตของฟังก์ชันที่สร้างขึ้นมา ก็สามารถทำได้โดยใช้คำสั่ง `intg` ซึ่งมีรูปแบบการใช้งานคือ

```
I = intg(a, b, f)
```

เมื่อ `I` คือผลลัพธ์ที่ได้จากการหาปริพันธ์, `a` และ `b` คือค่าเริ่มต้นและค่าสุดท้ายของการหาปริพันธ์, และ `f` คือฟังก์ชันแบบตัวแปรเดียวที่ต้องการหาปริพันธ์ พิจารณาตัวอย่างต่อไปนี้จะได้เข้าใจลักษณะการใช้งานของคำสั่ง `integrate` และ `intg` มากขึ้น

ตัวอย่างที่ 5 จงหาค่าปริพันธ์ของฟังก์ชัน $f(x)=2x+1$ สำหรับ $x=0$ ถึง 3



ในที่นี้การหาปริพันธ์ คือ การหาพื้นที่ใต้กราฟของสี่เหลี่ยมคางหมูตามที่แรเงา ดังนั้น

พื้นที่ส่วนที่แรเงา $=\frac{1}{2}(3)(1+7)=12$

วิธีทำ จากโจทย์จะได้ว่า

$$\int_{x=0}^{3} f(x)\,dx=\int_{x=0}^{3}(2x+1)\,dx$$

$$=\left(x^2+x\right)\Big|_{x=0}^{3}=\left(3^2+3\right)-\left(0-0\right)=12$$

ผู้ใช้สามารถตรวจคำตอบนี้ได้จากการใช้คำสั่งดังนี้

```
-->y = integrate('2*x+1', 'x', 0, 3)
 y  =
    12.
```

หรือใช้คำสั่ง

```
-->function z = f(x), z = 2*x+1, endfunction
-->I = intg(0, 3, f)
 I  =
    12.                                  //ให้ผลลัพธ์เท่ากับการใช้คำสั่ง integrate
```

ซึ่งให้ผลลัพธ์เท่ากัน

ตัวอย่างที่ 6 จงหาค่าของฟังก์ชัน $\int_0^3 \left(4-x^2\right) dx$

วิธีทำ จากโจทย์จะได้ว่า

$$\int_0^3 \left(4-x^2\right)dx = \left(4x-\frac{x^3}{3}\right)\Big|_0^3 = \left(4(3)-\frac{(3)^3}{3}\right)-(0-0) = 12-9 = 3$$

ผู้ใช้สามารถตรวจคำตอบนี้ได้จากการใช้คำสั่ง

```
-->y = integrate('4-x^2', 'x', 0, 3)
 y  =
    3.
```

หรือใช้คำสั่ง

```
-->function z = f(x), z = 4-x^2, endfunction
-->I = intg(0, 3, f)
 I  =
    3.                                   //ให้ผลลัพธ์เท่ากับการใช้คำสั่ง integrate
```

ซึ่งให้ผลลัพธ์เท่ากัน

ตัวอย่างที่ 7 จงหาค่าของฟังก์ชัน $\int_{x=5}^{9} \cosh\left(\frac{1}{4}x-1\right)dx$

วิธีทำ ถ้ากำหนดให้ $u=\frac{1}{4}x-1$ จะได้ว่า $du=\frac{1}{4}dx$ และ $dx=4du$

ลิมิตของปริพันธ์จำกัดเขตเป็นลิมิตของ x โดยที่ $u=\frac{1}{4}x-1$

เพราะฉะนั้นจะต้องเปลี่ยนลิมิตเป็นดังนี้

ลิมิตล่าง $x=5$ เปลี่ยนเป็นลิมิตล่าง $u=\frac{1}{4}(5)-1=\frac{1}{4}$

และลิมิตบน $x=9$ เปลี่ยนเป็นลิมิตบน $u=\frac{1}{4}(9)-1=\frac{5}{4}$

ดังนั้น

$$\int_{x=5}^{9} \cosh\left(\frac{1}{4}x-1\right)dx = 4\int_{u=1/4}^{5/4} \cosh(u)\,du = 4\sinh(u)\Big|_{1/4}^{5/4}$$

$$= 4\left[\sinh(5/4)-\sinh(1/4)\right] \approx 5.4$$

ผู้ใช้สามารถตรวจคำตอบนี้ได้จากการใช้คำสั่ง

```
-->y = integrate('cosh(x/4 - 1)', 'x', 5, 9)
 y  =
    5.3972271
```

หรือใช้คำสั่ง

```
-->function z = f(x), z = cosh(x/4 - 1), endfunction
-->I = intg(5, 9, f)
 I  =
    5.3972271                          //ให้ผลลัพธ์เท่ากับการใช้คำสั่ง integrate
```

ซึ่งให้ผลลัพธ์เท่ากัน

9.6 การแก้สมการอนุพันธ์อันดับหนึ่ง

ในส่วนนี้จะแสดงการใช้โปรแกรม SCILAB ในการแก้สมการอนุพันธ์อันดับหนึ่งโดยใช้คำสั่ง `ode` ซึ่งมีรูปแบบการเรียกใช้งานดังนี้

```
y = ode(y0, x0, x, f)
```

เมื่อ `y0` คือเวกเตอร์เลขจำนวนจริงของเงื่อนไขเริ่มต้น (initial conditions) ของฟังก์ชัน `y(x)` เมื่อ `x = 0`, `x0` คือค่าเริ่มต้นของตัวแปร `x`, `x = [x(1), x(2), x(3), ...]` คือเวกเตอร์เลขจำนวนจริงที่ใช้คำนวณหาค่าของฟังก์ชัน `y(x)`, `f` คืออนุพันธ์อันดับหนึ่งของฟังก์ชัน `y(x)`, และ `y` คือผลเฉลยที่ได้จากการแก้สมการอนุพันธ์อันดับหนึ่งซึ่งจะอยู่ในรูปของเวกเตอร์ `y = [y(x(1)), y(x(2)), y(x(3)), ...]` ตัวอย่างการใช้งานเช่น

พิจารณาฟังก์ชันต่อไปนี้

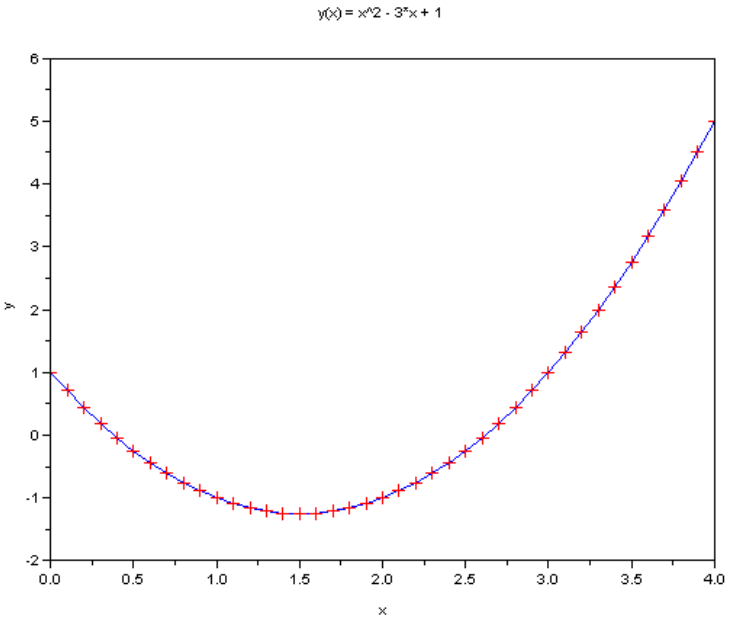
$$y(x) = x^2 - 3x + 1 \tag{1}$$

สังเกตว่าเมื่อ `x = 0` จะได้ว่า `y(x) = 1` โดยที่อนุพันธ์ของสมการ (1) คือ

$$ydot = \frac{dy}{dx} = 2x - 3 \tag{2}$$

ถ้าต้องการแก้สมการอนุพันธ์อันดับหนึ่งของสมการ (2) เพื่อให้ได้ผลลัพธ์เป็นสมการ (1) ก็สามารถทำได้โดยใช้ชุดคำสั่งดังนี้

```
-->function ydot = f(x, y), ydot = 2*x - 3, endfunction
-->y0 = 1;  x0 = 0;  x = 0:0.1:4;
-->y = ode(y0, x0, x, f);
-->plot(x, y);                    //วาดรูปกราฟของฟังก์ชันที่ได้จากการแก้สมการอนุพันธ์
-->yx = x^2 - 3*x + 1;
-->plot(yx, z, 'r+');             //วาดรูปกราฟของสมการ (1) เป็นเครื่องหมาย '+' สีแดง
-->xtitle('y(x) = x^2 - 3*x + 1', 'x', 'y');
```



รูปที่ 9.8 รูปกราฟของฟังก์ชัน `y(x)` (เส้นทึบ) และรูปกราฟที่ได้จากการแก้สมการอนุพันธ์ (เครื่องหมาย `'+'`)

ผลลัพธ์ที่ได้แสดงในรูปที่ 9.8 ซึ่งแสดงให้เห็นว่า ผลลัพธ์ที่ได้จากการแก้สมการอนุพันธ์ (รูปกราฟที่แสดงเฉพาะเครื่องหมาย `'+'` สีแดง) มีค่าเท่ากับรูปกราฟของฟังก์ชัน `y(x)` ในสมการ (1) ซึ่งเป็นกราฟเส้นทึบ

หมายเหตุ การแก้สมการอนุพันธ์ `dy/dx` ด้วยคำสั่ง `ode` จะให้ผลลัพธ์เป็นค่าของฟังก์ชัน `y(x)` เมื่อกำหนดค่าตัวแปร `x` มาให้ **ไม่ได้**ให้ผลลัพธ์เป็นสมการของฟังก์ชัน `y(x)`

ตัวอย่างที่ 8 จงหาคำตอบของสมการ $\frac{dy}{dx} + y = x$ (เงื่อนไขเริ่มต้นคือ `x = 0` และ `y = 1`) สำหรับ `x = 0.2, 0.4, 0.6, 0.8,` และ `1`

วิธีทำ จากการแก้สมการอนุพันธ์ที่กำหนดให้มานี้จะได้ผลลัพธ์ คือ

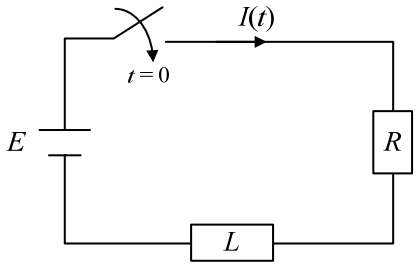
$$y = x + 2e^{-x} - 1$$

ดังนั้นผู้ใช้สามารถหาคำตอบของสมการอนุพันธ์โดยใช้คำสั่ง ode ได้ดังนี้

```
-->function ydot = f(x, y), ydot = x - y, endfunction
-->y0 = 1;  x0 = 0;  x = [0.2 0.4 0.6 0.8 1];
-->y = ode(y0, x0, x, f)            //ผลลัพธ์จากการแก้สมการอนุพันธ์ด้วยคำสั่ง ode
 y  =
    0.8374616    0.7406399    0.6976234    0.6986580    0.7357590
-->z = x + 2*exp(-x) - 1            //ผลลัพธ์จากการสมการผลเฉลย
 z  =
    0.8374615    0.7406401    0.6976233    0.6986579    0.7357589
```

จะเห็นได้ว่าผลลัพธ์ที่ได้จากการแก้สมการอนุพันธ์ด้วยคำสั่ง ode มีค่าเท่ากับผลลัพธ์จากการสมการผลเฉลย $y = x + 2e^{-x} - 1$

ตัวอย่างที่ 9 พิจารณาวงจรไฟฟ้า *RL* ต่อไปนี้



เมื่อกำหนดให้ค่าการเหนี่ยวนำตนเอง L = 5 H, ความต้านทาน R = 10 โอห์ม, และแหล่งจ่ายไฟ E = 15 โวลต์ โดยก่อนที่จะมีการเชื่อมต่อสวิทช์จะไม่มีกระแสไฟฟ้าไหลในวงจร จงคำนวณหาค่ากระแสไฟฟ้า `I(t)` เมื่อเวลา `t = 0.5, 1, 2,` และ `4` วินาที หลังจากทำการเชื่อมต่อสวิทช์

วิธีทำ จากกฎของเคอร์ชอฟจะได้สมการแรงดันไฟฟ้าดังนี้

$$L\frac{dI}{dt} + RI = E$$

โดยที่ I = 0 เมื่อ t = 0 จากการแก้สมการอนุพันธ์นี้จะได้ผลลัพธ์ คือ

$$I = \frac{E}{R}\left(1 - e^{-Rt/L}\right)$$

ดังนั้นกระแสไฟฟ้าเมื่อเวลา t = 0.5, 1, 2, และ 4 วินาที สามารถหาได้จากการใช้ชุดคำสั่งดังนี้

```
-->L = 10;  R = 20;  E = 12;
-->I0 = 0;  t0 = 0;  t = [0.5 1 2 4];
-->function  Idot = f(t, I), Idot = (E - R*I)/L, endfunction
-->I = ode(I0, t0, t, f)          //ผลลัพธ์จากการแก้สมการอนุพันธ์ด้วยคำสั่ง ode
 I  =
    0.3792723    0.5187989    0.5890106    0.5997987
-->z = E/R*(1 - exp(-R*t/L))      //ผลลัพธ์จากการสมการผลเฉลย
 z  =
    0.3792723    0.5187988    0.5890106    0.5997987
```

จะเห็นได้ว่าผลลัพธ์ที่ได้จากการแก้สมการอนุพันธ์ด้วยคำสั่ง ode มีค่าเท่ากับผลลัพธ์จากการสมการผลเฉลย $I = \frac{E}{R}\left(1 - e^{-Rt/L}\right)$

9.6 สรุป

บทนี้ได้แสดงให้เห็นว่าโปรแกรม SCILAB สามารถนำมาประยุกต์ใช้งานในการแก้ไขปัญหาทางคณิตศาสตร์ได้หลายอย่าง เช่น การหาตัวคูณร่วมน้อยและตัวหารร่วมมาก, การแยกตัวประกอบของพหุนาม, การลดรูปพหุนาม, การหาปริพันธ์จำกัดเขต, และการแก้สมการอนุพันธ์อันดับหนึ่ง เป็นต้น เพื่อเป็นแนวทางให้ผู้อ่านได้เห็นถึงความสามารถของโปรแกรม SCILAB และสามารถนำไปประยุกต์ใช้ในการเรียนการสอนและในงานวิจัยได้อย่างมีประสิทธิภาพ

นอกจากที่กล่าวมาในบทนี้แล้ว โปรแกรม SCILAB ยังสามารถนำไปประยุกต์ใช้งานในลักษณะอื่นๆ ได้อีก เช่น การประมาณค่าในช่วง (interpolation) และการแทนกลุ่มข้อมูลด้วยสมการเส้นที่เหมาะสม (data fitting), การวิเคราะห์ด้านการประมวลผลสัญญาณ (signal processing analysis), การวิเคราะห์ด้านระบบควบคุม (control system analysis), และการวิเคราะห์ด้านการสื่อสารดิจิทัล (digital communication analysis) เป็นต้น ผู้สนใจสามารถศึกษารายละเอียดเพิ่มเติมของกล่องเครื่องมือ (toolbox) ที่ใช้กับงานเฉพาะทางเหล่านี้ได้จาก http://www.scilab.org โดยผู้เขียนหวังว่าจะได้มีโอกาสเขียนหนังสือเกี่ยวกับการประยุกต์ใช้งานเหล่านี้ในอนาคต

9.7 แบบฝึกหัดท้ายบท

9.1 จงอธิบายประโยชน์ของเวกเตอร์และเมทริกซ์ตรรกะ พร้อมแสดงตัวอย่างการใช้งาน

9.2 จงเขียนโปรแกรมเพื่อทำ symbolic triangularization ของเมทริกซ์ขนาด 2×2

9.3 จงหาตัวคูณร่วมน้อยและตัวหารร่วมมากของ

9.3.1) เลขจำนวนเต็ม 12 และ 42

9.3.2) เลขจำนวนเต็ม 16, 39, และ 81

9.3.3) พหุนาม $x^2 - 2x - 3$ และ $x^2 - x - 2$

9.3.4) พหุนาม $3x^2 - 9x + 6$ และ $3x^3 - 7x^2 + 4$

9.3.5) พหุนาม $x^2 - 2x + 1$, $x^3 - 3x + 2$, และ $x^3 - 3x^2 + 3x - 1$

9.3.6) พหุนาม $x^2 + 3x + 2$, $x^3 + 6x^2 + 11x + 6$, และ $x^4 + 10x^3 + 35x^2 + 50x + 24$

9.4 จงหาแยกตัวประกอบของพหุนามต่อไปนี้

9.4.1) $2x^2 - 4x + 2$

9.4.2) $4x^2 - 4x + 1$

9.4.3) $3x^3 - 4.5x^2 - 4.5x + 3$

9.4.4) $6x^3 - 15x^2 + 3x + 6$

9.4.5) $2x^4 - 7x^3 + x^2 + 16x - 12$

9.4.6) $x^4 - 6x^3 + 3x^2 + 26x - 24$

9.5 จงคำนวณหาค่าต่อไปนี้

9.5.1) $2y^3 + 5y^2 + y - 2$ หารด้วย $y + 1$

9.5.2) $y^2 - 5y + 10$ หารด้วย $y^2 - 4$

9.5.3) $y^3 - 27$ หารด้วย $y^2 - 9$

9.6 จงหาพื้นที่ใต้กราฟของฟังก์ชันต่อไปนี้

9.6.1) $y = 3x - 1$ สำหรับ $1 \le x \le 5$

9.6.2) $y = x^2$ สำหรับ $-3 \le x \le 3$

9.7 จงหาปริพันธ์จำกัดเขตดังต่อไปนี้

9.7.1) $\int_1^2 (2x+5)dx$

9.7.2) $\int_0^3 (4+x^3)dx$

9.7.3) $\int_0^1 \left(x^2+\sqrt{x}\right)dx$

9.7.4) $\int_0^{\pi} \sin(x)\,dx$

9.7.5) $\int_{-\pi/2}^{\pi/2} (8y^2+\sin y)dy$

9.7.6) $\int_{-1}^{1} (x+1)^2 dx$

9.7.7) $\int_0^{\pi/2} \cos(x)\sin(x)\,dx$

9.8 กำหนดให้สมการอนุพันธ์ $\frac{dy}{dt} = y^2 - y\sin(t) + \cos(t)$ จงหาค่า $y(t)$ ที่เวลา $t = \pi/4, \pi/2,$ และ π เมื่อกำหนดเงื่อนไขเริ่มต้น $y(0) = 0$

ภาคผนวก ก

ตัวอย่างการทำ symbolic triangularization

ในส่วนนี้จะยกตัวอย่างการคำนวณเชิงสัญลักษณ์สำหรับการทำสามเหลี่ยมบนของเมทริกซ์สายอักขระ หรือที่เรียกกันว่า “symbolic triangularization” ตามที่ได้ยกตัวอย่างไว้ในหัวข้อที่ 2.3 กล่าวคือ ถ้ากำหนดให้เมทริกซ์ **X** มีค่าเท่ากับ

$$\mathbf{X} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{ก1}$$

โดยที่ `a`, `b`, `c`, และ `d` คือตัวแปรค่าคงที่ใดๆ และเมื่อทำ symbolic triangularization ของเมทริกซ์ **X** โดยใช้คำสั่ง `trianfml(X)` จะได้ผลลัพธ์เป็น

$$\mathbf{Y} = \begin{bmatrix} c & d \\ 0 & cb - ad \end{bmatrix} \tag{ก2}$$

ขั้นตอนต่อไปจะอธิบายการทำ symbolic triangularization เพื่อเปลี่ยนเมทริกซ์ **X** ให้เป็นเมทริกซ์ **Y** โดยใช้เทคนิคการดำเนินการตามแถวขั้นมูลฐาน (elementary row operation) ดังนี้ จากสมการ (ก1) สามารถที่จะสมมติได้ว่า เมทริกซ์ **X** ประกอบไปด้วย 2 สมการ คือ

$$cZ_1 + dZ_2 = 0 \tag{ก3}$$

$$aZ_1 + bZ_2 = 0 \tag{ก4}$$

โดยที่ตัวแปร Z_1 และ Z_2 เป็นตัวแปรชั่วคราวที่ถูกกำหนดขึ้นเพื่อใช้ในการสร้างสมการสองตัวแปรจากคุณสมบัติการดำเนินการตามแถวขั้นมูลฐาน สมการ (ก4) สามารถที่จะแทนได้ด้วยสมการที่เป็นผลลัพธ์จากการนำตัวแปรค่าคงที่ c คูณกับสมการ (ก4) แล้วลบด้วยผลลัพธ์ที่ได้จากการนำตัวแปรค่าคงที่ a คูณกับสมการ (ก3) นั่นคือ

$$c\,(ก4) - a\,(ก3) = 0$$

$$c\left(aZ_1 + bZ_2\right) - a\left(cZ_1 + dZ_2\right) = 0$$

$$0(Z_1) + (cb - ad)Z_2 = 0 \qquad (ก5)$$

เพราะฉะนั้นเมื่อนำค่าสัมประสิทธิ์ของสมการ (ก3) และ (ก5) มาจัดรูปให้อยู่ในรูปของเมทริกซ์ก็จะได้ดังนี้

$$\underbrace{\begin{bmatrix} c & d \\ 0 & cb-ad \end{bmatrix}}_{\mathbf{Y}} \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

ซึ่งก็คือเมทริกซ์ **Y** ที่สอดคล้องกับสมการ (ก2) ตามที่ต้องการ

ภาคผนวก ข

รหัสข้อผิดพลาด

ในส่วนนี้จะขอแสดงรายการของรหัสข้อผิดพลาด (error list) ของโปรแกรม SCILAB เพื่อใช้เป็นข้อมูลอ้างอิงให้นักพัฒนาโปรแกรมสามารถเข้าใจว่ารหัสข้อผิดพลาดมีอะไรบ้าง และเลขรหัสแต่ละหมายเลขหมายถึงอะไร ดังแสดงในตารางต่อไปนี้

เลขรหัส	คำอธิบาย
1	incorrect assignment
2	invalid factor
3	waiting for right parenthesis
4	undefined variable : %s
5	inconsistent column/row dimensions
6	inconsistent row/column dimensions
7	dot cannot be used as modifier for this operator
8	inconsistent addition
9	inconsistent subtraction
10	inconsistent multiplication
11	inconsistent right division
12	inconsistent left division
13	redefining permanent variable
14	eye variable undefined in this context
15	submatrix incorrectly defined
16	incorrect command!
17	stack size exceeded! (Use stacksize function to increase it)
18	too many variables!
19	Problem is singular
20	%dth argument must be square matrix
21	invalid index
22	recursion problems. Sorry....
23	Matrix norms available are 1, 2, inf, and fro
24	convergence problem...
25	bad call to primitive :%s

เลขรหัส	คำอธิบาย
26	too complex recursion! (recursion tables are full)
27	division by zero...
28	empty function...
29	matrix is not positive definite
30	invalid exponent
31	incorrect string
32	singularity of log or tan function
33	too many "":""
34	incorrect control intruction syntax
35	Syntax error in an if,a while or a select instruction
36	%dth argument is incorrect here
37	incorrect function at line %d
38	file name incorrect
39	incorrect number of arguments
40	waiting for end of command
41	incompatible LHS
42	incompatible RHS
43	not implemented in scilab....
44	%dth argument is incorrect
45	null matrix (argument # %d)
46	incorrect syntax
47	end or else is missing...
48	input line longer than buffer size %d
49	incorrect file or format
50	subroutine not found : %s
52	%dth argument must be a real matrix
53	%dth input is invalid (waiting for real or complex matrix)
54	%dth argument type must be polynomial
55	%dth argument type must be character string
56	%dth argument must be a list
57	problem with comparison symbol...
58	incorrect number of arguments in function call...
59	incorrect # of outputs in the function
60	argument with incompatible dimensions
61	direct acces file : give format
62	end of file at line %d
63	%d graphic terminal?
64	integration fails
65	%d: logical unit already used
66	no more logical units available!
67	unknown file format
69	floating point exception
70	too many arguments in fort (max 30)
71	this variable is not valid in fort

เลขรหัส	คำอธิบาย
72	%s is not valid in this context
73	error while linking
74	Leading coefficient is zero
75	Too high degree (max 100)
76	for x = val with type(val) = %d is not implemented in Scilab
77	%s : wrong number of rhs arguments
78	%s : wrong number of lhs arguments
80	incorrect function (argument n:%s)
81	Argument %d of %s: wrong type argument, expecting a real or complex matrix
82	Argument %d of %s: wrong type argument, expecting a real matrix
83	Argument %d of %s: wrong type argument, expecting a real vector
84	Argument %d of %s: wrong type argument, expecting a scalar
85	host does not answer...
86	uncontrollable system
87	unobservable system
88	sfact : singular or assymetric problem
89	%dth argument has incorrect dimensions
90	%dth argument must be a transfer matrix
91	%dth argument must be in state space form
92	%dth argument must be a rational matrix
93	%dth argument must be in continuous time
94	%dth argument must be in discrete time
95	%dth argument must be SISO
96	time domain of %dth argument is not defined
97	%dth argument must be a system in state space or transfer matrix form
98	variable returned by scilab argument function is incorrect
99	elements of %dth argument must be in increasing order!
100	the elements of %dth argument are not in (strictly) decreasing order
101	last element of %dth argument is not equal to the first
102	variable or function %s is not in file %s
103	variable %s is not a valid rational function
104	variable %s is not a valid state space representation
105	undefined fonction
106	function name already used
107	too many functions are defined (maximum #:%d)
108	too complex for scilab, may be a too long control instruction
109	too large, can't be displayed
110	%s was a function when compiled but is now a primitive!
111	trying to re-define function %s
112	Cannot allocate more memory
113	too large string
114	too many linked entry points
115	Stack problem detected within a loop
116	%dth argument has incorrect value

เลขรหัส	คำอธิบาย
117	list element number %d is Undefined
118	%dth argument must be a named variable not an expression
119	indices for non-zero elements must be given by a 2 column matrix
121	incompatible indices for non-zero elements
122	logical unit number should be larger than %d
123	fonction not bounded from below
124	problem may be unbounded :too high distance between two consecutive iterations
126	Inconsistent constraints
127	no feasible solution
128	degenerate starting point
129	no feasible point has been found
130	optimization fails: back to initial point
131	optim: stop requested by simulator (ind=0)
132	optim: incorrect input parameters
133	too small memory
134	optim: problem with initial constants in simul
135	optim : bounds and initial guess are incompatible
136	optim : this method is NOT implemented
137	NO hot restart available in this method
138	optim : incorrect stopping parameters
139	optim : incorrect bounds
141	incorrect function (argument n:%d)
142	hot restart : dimension of working table (argument n:%d)
143	optim : df0 must be positive !
144	Undefined operation for the given operands check or define function %s for overloading
201	Argument %d of %s: wrong type argument, expecting a real or complex matrix
202	Argument %d of %s: wrong type argument, expecting a real matrix
203	Argument %d of %s : wrong type argument, expecting a real vector
204	Argument %d, wrong type argument: expecting a scalar
205	Argument %d of %s: wrong matrix size (%d) expected
206	Argument %d of %s: wrong vector size (%d) expected
207	Argument %d of %s: wrong type argument, expecting a matrix of strings
208	Argument %d of %s: wrong type argument, expecting a booleen matrix
209	Argument %d of %s: wrong type argument, expecting a matrix
210	Argument %d of %s: wrong type argument, expecting a list
211	Argument %d of %s: wrong type argument, expecting a function or string (external function)
212	Argument %d of %s: wrong type argument, expecting a polynomial matrix
213	Argument %d of %s: wrong type argument, expecting a working integer matrix
214	Argument %d of %s: wrong type argument, expecting a vector
215	%dth argument type must be boolean
216	%dth argument type must be boolean or scalar
217	%dth argument must be a sparse matrix of scalars
218	%dth argument must be a handle to sparse lu factors

เลขรหัส	คำอธิบาย
219	%dth argument must be a sparse or full scalar matrix
220	null variable cannot be used here
221	A sparse matrix entry is defined with two differents values
222	lusolve not yet implemented for full RHS
223	It is not possible to redefine the %s primitive this way (see clearfun).
224	Type data base is full
225	This data type is already defined
226	Inequality comparison with empty matrix
227	Missing index
228	reference to the cleared global variable %s
230	semidef fails
231	First argument must be a single string
232	Entry name not found
233	Maximum number of dynamic interfaces reached
234	link: expecting more than one argument
235	link: problem with one of the entry point
236	link: the shared archive was not loaded
237	link: Only one entry point allowed On this operating system
238	link: First argument cannot be a number
239	You cannot link more functions, maxentry reached
240	File %s already exists or directory write access denied
241	File %s does not exist or read access denied
242	binary direct acces files must be opened by ""file""
243	C file logical unit not allowed here
244	Fortran file logical unit not allowed here
245	No input file associated to logical unit %d
246	function not defined for given argument type(s)
248	%dth argument is not a valid variable name
249	%dth argument must not be an empty string
250	Recursive extraction is not valid in this context
251	bvode: ipar dimensioned at least 11
252	bvode: ltol must be of size ipar(4)
253	bvode: fixpnt must be of size ipar(11)
254	bvode: ncomp must be less than 20
255	bvode: m must be of size ncomp
256	bvode: sum(m) must be less than 40
257	bvode: sum(m) must be less than 40
258	bvode: input data error
259	bvode: no. of subintervals exceeds storage
260	bvode: Th colocation matrix is singular
261	Interface property table is full
262	too many global variables!,max number is %d
263	Error while writing in file,(disk full or deleted file)
264	%dth argument must not contain NaN or Inf

เลขรหัส	คำอธิบาย
265	A and B must have equal number of rows
266	A and B must have equal number of columns
267	A and B must have equal dimensions
268	invalid return value for function passed in argument %d
269	%dth argument eigenvalues must have negative real parts
270	%dth argument eigenvalues modulus must be less than one
271	Size varying argument aeye(), (arg %d) not allowed here
272	endfunction is missing
273	Instruction left hand side: waiting for a dot or a left parenthesis
274	Instruction left hand side: waiting for a name
275	varargout keyword cannot be used here
276	Missing operator, comma, or semicolon
277	Too many commands defined

บรรณานุกรม

[1] Stephen J. Chapman, *MATLAB Programming for Engineers*, 2nd Edition, Thomson-Engineering, 2001.

[2] Gilberto E. Urroz, Elementary Mathematical Functions in SCILAB, September 2002.

[3] Gilberto E. Urroz, Programming with SCILAB, September 2002.

[4] Gilberto E. Urroz, Input/Output and File Handling Functions in SCILAB, September 2001.

[5] Gilberto E. Urroz, SCILAB: A Free Software for Scientific Calculations Part V: Breakpoints and Debugging in SCILAB, September 2002.

[6] Gilberto E. Urroz, Some Symbolic Operations in SCILAB, September 2002.

[7] Introduction to Scilab, Scilab Group, 1994

[8] Retrieved from http://www.scilab.org

[9] Retrieved from http://www.engineering.usu.edu/cee/faculty/gurro/Scilab.html

[10] ลัญจกร วุฒิสิทธิกุลกิจ, *MATLAB การประยุกต์ใช้งานทางวิศวกรรมไฟฟ้า*, สำนักพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย, พ.ศ. 2547.

[11] สุธรรม ศรีเกษม, เมธินทร์ ทรงชัยกุล, และ สง่า ศรีศุภปรีดา, *MATLAB เพื่อการแก้ปัญหาทางวิศวกรรม*, สำนักพิมพ์มหาวิทยาลัยรังสิต, พ.ศ. 2521.

ประวัติผู้เขียน

ผศ.ดร.ปิยะ โควินท์ทวีวัฒน์

URL: http://home.npru.ac.th/~t3058

การศึกษา

- พ.ศ. 2537 ปริญญาตรี (เกียรตินิยม) Thammasat University, Bangkok, Thailand
- พ.ศ. 2541 ปริญญาโท Chalmers University of Technology, Göteborg, Sweden
- พ.ศ. 2547 ปริญญาเอก Georgia Institute of Technology, Atlanta, GA, USA

ประสบการณ์ทำงาน

- อาจารย์และนักวิจัย มหาวิทยาลัยราชภัฏนครปฐม (พ.ศ. 2548 – ปัจจุบัน)
- Technical Intern, Channels Department, Seagate Research Center, Pittsburgh, USA (รวม 1 ปี)
- ผู้ช่วยนักวิจัย ห้องปฏิบัติการเทคโนโลยีเครือข่าย ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (เนคเทค) ประเทศไทย (1 ปี)
- วิศวกร บริษัท TT&T (มหาชน) จำกัด ประเทศไทย (3.5 ปี)

ผลงานทางวิชาการ

- วารสารตีพิมพ์นานาชาติ 18 ฉบับ
- สิทธิบัตร: P. Kovintavewat, J. R. Barry, F. M. Erden, and E. M. Kurtas, “Method and Apparatus for Providing Iterative Timing Recovery,” invention disclosure filed by Seagate Technology, patent pending, September 2004.

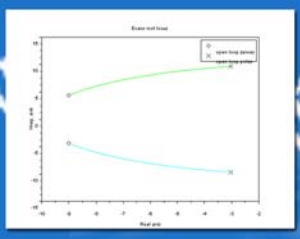
- หนังสือ
  - ปิยะ โควินท์ทวีวัฒน์, *การประมวลผลสัญญาณสำหรับการจัดเก็บข้อมูลดิจิทัล เล่ม 1 : พื้นฐานช่องสัญญาณอ่าน-เขียน*, ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (เนคเทค), พ.ศ. 2550.
  - ปิยะ โควินท์ทวีวัฒน์, *การประมวลผลสัญญาณสำหรับการจัดเก็บข้อมูลดิจิทัล เล่ม 2 : การออกแบบวงจรภาครับ*, ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (เนคเทค), พ.ศ. 2550.
  - ปิยะ โควินท์ทวีวัฒน์, *คู่มือโปรแกรมภาษา SCILAB สำหรับผู้เริ่มต้น* (พิมพ์ครั้งที่ 2), ศูนย์ผลิตตำราเรียน, สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ, พ.ศ. 2549.
  - B. Vasic and E. M. Kurtas, *Coding and Signal Processing for Recording Systems*, Book Chapter: *Interpolated timing recovery*, CRC press, pp. 27-1 – 27-16, 2005.

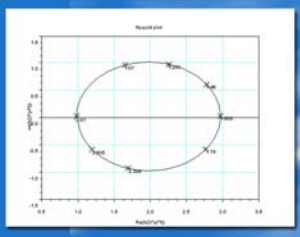
งานสำคัญ

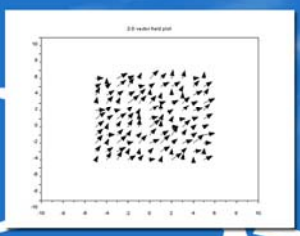
- ผู้ช่วยอธิการบดีฝ่ายวิจัย
- ประธานโปรแกรมวิศวกรรมโทรคมนาคม
- บรรณาธิการ วารสารวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏนครปฐม

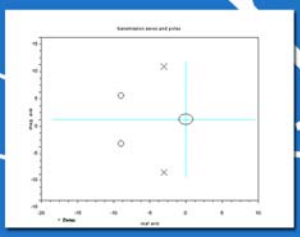
งานวิจัย

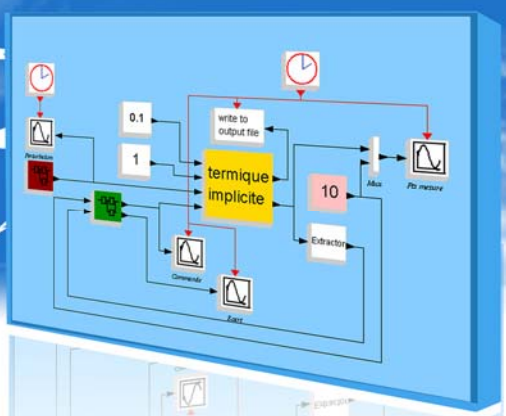
- ระบบการประมวลผลสัญญาณสำหรับการจัดเก็บข้อมูลดิจิทัล
- เทคโนโลยีการเข้า-ถอดรหัสแก้ไขข้อผิดพลาดของช่องสัญญาณ
- เทคโนโลยีการตรวจหาแบบวนซ้ำ (iterative detection)
- เทคโนโลยีสมาร์ตการ์ด (smart card)
- เทคโนโลยี RFID (radio frequency identification)











ผศ.ดร.ปิยะ โควินท์ทวีวัฒน์

คณะวิยาศาสตร์และเทคโนโลยี

สงวนลิขสิทธิ์ มหาวิทยาลัยราชภัฏนครปฐม

85 ถนนมาลัยแมน อำเภอเมือง จังหวัดนครปฐม 73000

โทร. 034-261021 โทรสาร 034-261065