โปรแกรมเครื่องคิดเลขสำหรับการจูนพีไอดี

CALCULATOR PROGRAMMING FOR PID TUNING

โดย

นาย นัทธพงศ์ วีระวงศ์

นาย เขมจิรา อัศวธนาธร

นาย อิทธินันท์ เลาหอภิชาญโชติ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ ปีการศึกษา 2550

ภาควิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมเครื่องคิดเลขสำหรับการจูนพีไอดี

CALCULATOR PROGRAMMING FOR PID TUNING

ผู้จัดทำ	1. นาย นัทธพงศ์	วีระวงศ์	47010377
	2. นาย เขมจิรา	อัศวธนาธร	47010789
	3. นาย อิทธินันท์	เลาหอภิชาญโชติ	47010993

..... อาจารย์ที่ปรึกษา

(ผศ.ดร.ปรเมษฐ์ ประนยานันทน์)

โปรแกรมเครื่องคิดเลขสำหรับการจูนพีไอดี

โดย	นาย นัทธพงศ์	วีระวงศ์	47010377
	นาย เขมจิรา	อัศวธนาธร	47010789
	นาย อิทธินันท์	เลาหอภิชาญโชติ	47010993

อาจารย์ที่ปรึกษา
ผศ.ดร. ปรเมษฐ์ ประนยานันทน์

บทคัดย่อ

ปริญญานิพนธ์นี้ นำเสนอการออกแบบ และการใช้งานพร้อมทั้งแสดงผลการทำงานของโปรแกรม JKI Toolbox ซึ่งเป็นโปรแกรมที่สร้างขึ้นจากการเลียนแบบการทำงานของโปรแกรม MATLAB ซึ่งเป็นโปรแกรมที่ใช้ในการคำนวณทางคณิตศาสตร์จากคอมพิวเตอร์

สำหรับการออกแบบ เครื่องคิดเลข TI-89 เป็นเครื่องคิดเลขที่ สามารถเขียนโปรแกรมคำสั่งเพิ่มเติมเข้าไปในเครื่องได้ และยังง่ายในการพกพาไปในทุกที่ ซึ่งบางครั้งในการทำงาน เป็นการยากที่จะนำอุปกรณ์คำนวณขนาดใหญ่ ไปใช้งานได้ ดังนั้น TI-89 คือทางเลือกที่ดีทางหนึ่ง

ในการออกแบบโปรแกรมโดยรวม JKI Toolbox นี้จะมีฟังก์ชันในการทำงานหลายอย่าง ที่เป็นประโยชน์ในการคำนวณเกี่ยวกับทฤษฎีระบบควบคุม ซึ่งฟังก์ชันของโปรแกรมเขียนขึ้นจากคำสั่งพื้นฐานของเครื่องคิดเลข TI-89

Calculator Programming For PID Tuning

By Mr. Nattapong Weerawong 47010377
Mr. Khemjira Assawathanatorn 47010789
Mr. Itthinan Lauha-apichanchot 47010993

Advisor
Asst. Prof. Dr. Poramate Pranayanuntana

ABSTRACT

In this thesis, designing JKI Toolbox to using by TI-89 calculator. The JKI Toolbox is a program that use for control theory. The program is working as same as control toolbox is MATLAB program

For designing, TI-89 calculator is a smart calculator that can programming function. The calculator is easy to use and can carry to everywhere. For anywhere that hard to take a computer to use, This calculator is a chosen one

For the program overview, the program have many functions to using for control theory. For the function, designing with command code in calculator TI-89

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดี ก็เพราะได้รับการสนับสนุนจากบุคคลหลายท่าน โดยทางผู้จัดทำขอขอบพระคุณ ผศ.ดร.ปรเมษฐ์ ประนยานันทน์ ที่ให้ความกรุณาแนะนำที่เป็นประโยชน์ ตลอดจนให้ความเอาใจใส่ดูแลสอบถามความก้าวหน้าอย่างสม่ำเสมอ พร้อมทั้งคณาจารย์ทุกท่าน ที่ได้ประสิทธิประสาทความรู้ด้านต่างๆและให้คำปรึกษาแก่ผู้จัดทำ ขอขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้

ขอขอบคุณภาควิชา วิศวกรรมระบบควบคุม ที่มีเครื่องมือและอุปกรณ์ในการทำงานให้ใช้อย่างครบครัน ขอบคุณ พี่ๆธุรการที่ให้ความสะดวกในเรื่องต่างๆ

สุดท้ายนี้ผู้จัดทำขอกราบขอบพระคุณบิดา มารดา เพื่อนๆ ของผู้จัดทำ ที่ได้ให้การอุปการะมาโดยตลอด และให้กำลังใจแก่ผู้จัดทำตลอดเวลา

คณะผู้จัดทำ	นาย นัทธพงศ์	วีระวงศ์	47010377
	นาย เขมจิรา	อัศวธนาธร	47010789
	นาย อิทธินันท์	เลาหอภิชาญโชติ	47010993

สารบัญ

สารบัญ(ต่อ)

สารบัญภาพ

สารบัญภาพ(ต่อ)

สารบัญภาพ(ต่อ)

บทที่ 1
บทนำ

1.1 ที่มาของการทำงาน

ที่มาของการทำโปรเจค JKI Toolbox นี้เกิดจากการที่ได้ศึกษาการใช้งานโปรแกรมMATLAB ซึ่งเป็นโปรแกรมที่ใช้ในการคำนวณทางคณิตศาสตร์ จึงเกิดแนวคิดที่จะทำโปรแกรมที่มีคุณสมบัติแบบเดียวกัน แต่สามารถใช้งานได้ในเครื่องคำนวณที่มีขนาดเล็ก ที่มีความสะดวกต่อการพกพาและการนำไปใช้งานในสถานที่ปฏิบัติงานจริงเช่น ภายในโรงงานอุตสาหกรรม หรือตามสภาพแวดล้อมการทำงานที่จำกัด ซึ่งโปรแกรมMATLABต้องใช้งานกับเครื่องคอมพิวเตอร์ ซึ่งมีขนาดใหญ่ไม่เหมาะสมในการทำงาน และจากปัญหาที่ได้พบจริงในการทำงาน ด้านการคำนวณและการปรับแต่งระบบควบคุมแบบพีไอดี(PID) ภายในกระบวนการอุตสาหกรรม เช่น ในโรงงานอุตสาหกรรม หรือสภาพแวดล้อมในการทำงานที่จำกัด ที่ทำให้ไม่สามารถนำเครื่องคอมพิวเตอร์ไปใช้งานในด้านการปรับแต่งระบบควบคุมแบบพีไอดี(PID)ได้ จึงเกิดความคิดที่จะนำโปรแกรมที่สามารถคำนวณทางคณิตศาสตร์ ที่สามารถปรับแต่งระบบควบคุมแบบพีไอดี(PID) ผ่านการใช้งานทางเครื่องคำนวณที่มีขนาดเล็ก เหมาะสมที่จะทำงานในสภาพแวดล้อมที่จำกัดดังกล่าวเพื่อเพิ่มประสิทธิภาพ ในการทำงานให้ดียิ่งขึ้น และแก้ปัญหาที่เกิดขึ้นดังกล่าวให้หมดไป

1.2 วัตถุประสงค์

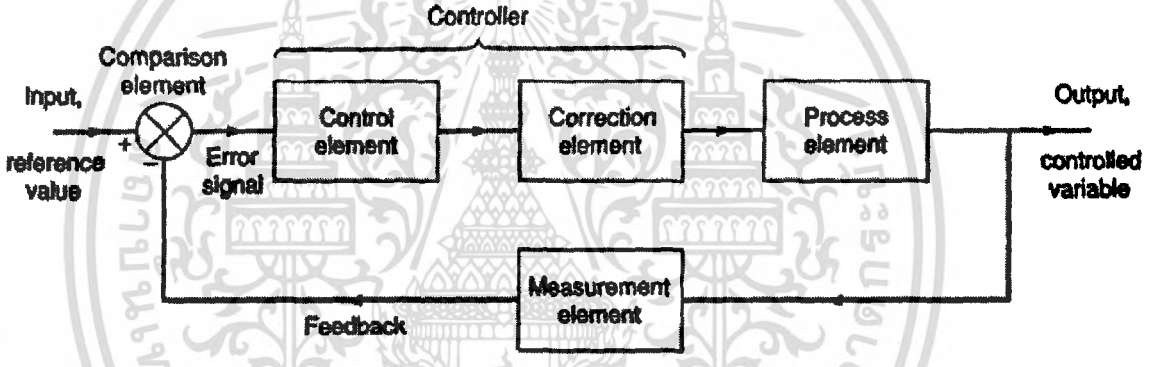
โปรเจค JKI Toolbox เป็นโปรเจคที่มีแนวคิดในการแก้ปัญหาที่จะเกิดขึ้นจริง ในการควบคุมระบบแบบพีไอดี(PID)ในอุตสาหกรรม ซึ่งในอุตสาหกรรมสมัยใหม่นั้น มีการใช้ระบบควบคุมแบบพีไอดี(PID)แพร่หลาย ไม่ว่าจะเป็นการควบคุมภายในโรงงานอุตสาหกรรม ในพื้นที่งาน(Plant)ที่มีการควบคุมเครื่องจักร หรือในระบบอุตสาหกรรมในรูปแบบอื่น ล้วนนิยมใช้การควบคุมระบบแบบพีไอดี(PID) แทบทั้งสิ้น เพราะสามารถสร้างเสถียรภาพที่ดีให้กับระบบ ทำให้ระบบเกิดการทำงานได้อย่างมีประสิทธิภาพ แต่ในการใช้งาน ระบบการควบคุมแบบพีไอดี(PID)นั้นมีความยุ่งยากที่เกิดขึ้นจากการคำนวณสมการทางคณิตศาสตร์ที่ซับซ้อน และมีตัวแปรในการคำนวณมากมาย จึงนิยมคำนวณสมการระบบควบคุมพีไอดี(PID)โดยใช้คอมพิวเตอร์ช่วยในการคำนวณ แต่เนื่องจากคอมพิวเตอร์มีตัวเครื่องที่มีขนาดใหญ่ ไม่เหมาะสมที่จะนำไปทำงานภายในโรงงานอุตสาหกรรม หรือพื้นที่งาน(Plant)ที่มีขนาดเล็ก พื้นที่จำกัด อาจเนื่องจาก การวางเครื่องจักร หรืออุปกรณ์ที่เป็นอุปสรรคต่อการทำงานของเครื่องคอมพิวเตอร์ จึงทำให้เกิดปัญหาในการปรับแต่งระบบควบคุมพีไอดี(PID)โดยใช้คอมพิวเตอร์ที่ต้องการ การปรับแต่งระบบควบคุมแบบพีไอดี(PID)ในโรงงาน หรือในพื้นที่งาน(Plant)ที่ต้องการ การแก้ไขในทันทีทันใด คอมพิวเตอร์จึงไม่เหมาะสมในการนำเข้าไปใช้งาน ทำให้เกิดปัญหาขึ้น โดยโปรเจคนี้จะเป็นการแก้ไขปัญหาเหล่านี้โดยใช้เครื่องคำนวณรุ่น TI-89 นี้จะมีระบบประมวลผลกลาง(CPU) ที่สูง สามารถนำโปรแกรมการคำนวณเข้ามาใช้งานผ่านทางเครื่องคำนวณรุ่น TI-89 ได้ ทำให้สามารถนำเครื่องคำนวณที่มีขนาดเล็ก พกพาสะดวก ง่ายต่อการทำงาน และนำไปใช้งานภายในโรงงานอุตสาหกรรม หรือในพื้นที่งาน(Plant)ได้อย่างมีประสิทธิภาพ แทนการใช้งานจากเครื่องคอมพิวเตอร์

โปรเจคนี้เป็นการสร้างโปรแกรม ด้านการคำนวณขึ้น คือ โปรแกรม JKI Toolbox ซึ้งประกอบไปด้วย ฟังก์ชันที่ถูกสร้างขึ้นเพื่อใช้ในการคำนวณทางคณิตศาสตร์ที่เกี่ยวข้องกับระบบควบคุมพีไอดี(PID)โดยสามารถนำโปรแกรมไปดาวน์โหลดใช้งานกับเครื่องคำนวณรุ่น TI-89 ได้ เพื่อนำไปใช้ในงานด้านการปรับแต่งระบบควบคุมแบบพีไอดี(PID) ต่อไป

บทที่ 2
หลักการและทฤษฎี

2.1 ทฤษฎีระบบควบคุมป้อนกลับ

ระบบควบคุมแบบปิดสามารถพิจารณาได้ว่า ประกอบด้วยระบบย่อยที่ต่อวางกันตามรูปที่ 2.1 ในความเป็นจริงระบบย่อยเหล่านี้ เราอาจไม่สามารถที่จะแยกแต่ละชิ้นส่วนออกมาเป็นส่วนๆ ได้ หรือแยกอุปกรณ์ใดอุปกรณ์หนึ่งอย่างชี้ชัดลงไปได้ว่า อุปกรณ์นั้นทำหน้าที่อย่างหนึ่งอย่างใดโดยเฉพาะ แต่อุปกรณ์ในความเป็นจริงเหล่านี้ สามารถแยกการทำงานออกเป็นส่วนต่างๆ ได้ตามที่แสดงในรูปที่ 2.1



รูปที่ 2.1 ระบบควบคุมแบบป้อนกลับ

โดยส่วนต่างๆ ในระบบควบคุมแบบปิดนี้จะประกอบด้วย

1. ส่วนเปรียบเทียบ (Comparison Element) ส่วนนี้จะทำหน้าที่เปรียบเทียบค่าตัวแปรที่เราต้องการออกมา หรืออาจเรียกว่าค่ามาตรฐานของตัวแปรที่เราต้องการ เพื่อเปรียบเทียบกับค่าที่เราวัดค่าตัวแปรนั้นได้ในสถาพความเป็นจริง ซึ่งเป็นค่าที่เป็นเอาท์พุทของระบบ ส่วนนี้จะให้สัญญาณหรือค่าความผิดพลาดออกมา ซึ่งความผิดพลาดนี้จะบอกให้ทราบว่าขณะนี้ค่าตัวแปรที่ต้องการควบคุมนั้นมีค่าอยู่แตกต่างจากค่าที่เราต้องการให้มันเป็นเท่าใด นั่นคือ

ความผิดพลาด = ค่าสัญญาณอ้างอิง - ค่าสัญญาณที่วัดได้

2. ส่วนควบคุม (Control Element) เป็นส่วนที่ทำหน้าที่ตัดสินใจว่าจะต้องทำอย่างไร เมื่อได้รับสัญญาณความผิดพลาด เรามักจะใช้คำว่าคอนโทรลเลอร์ (Controller) เมื่อเราเรียกส่วนนี้รวมถึง ส่วนชดเชย (Correction Element)

3. ส่วนชดเชย (Correction Element) ส่วนนี้มีหน้าที่ในการกำหนดการเปลี่ยนแปลง ของตัวแปร เพื่อที่จะลดค่าความผิดพลาดให้น้อยลง เรามักเรียกอุปกรณ์ที่ทำหน้าที่ในส่วนนี้ว่า แอกชวลเตอร์ (Actuator)

4. ส่วนกระบวนการ (Process Element) หรือพื้นที่งาน (plant) จะเป็นระบบซึ่งเราต้องการควบคุมค่าตัวแปรตัวใดตัวหนึ่งหรือหลายตัว

5. ส่วนวัดค่า (Measure Element) ส่วนนี้จะเป็นส่วนของเครื่องมือวัด ซึ่งเครื่องมือวัดนี้จะให้สัญญาณที่แสดงถึงขนาดของตัวแปรที่เราต้องการที่จะควบคุม และเมื่อได้ค่าที่วัดแล้ว ก็จะมีการป้อนสัญญาณนั้นกลับเข้าสู่ส่วนเปรียบเทียบ (Comparison Element) เพื่อให้ระบบพิจารณาว่ามีความผิดพลาดเกิดขึ้นหรือไม่

การทำงานของระบบป้อนกลับนี้ จะทำไปเรื่อยๆ จนกว่าค่ามาตรฐาน และค่าที่วัดได้มีค่าเท่ากัน นั่นคือระบบควบคุมของเรา สามารถที่จะควบคุม ให้ค่าตัวแปรที่เราต้องการ มีค่าตามที่เรากำหนดได้เรียบร้อยแล้วนั่นเอง ส่วนสำคัญและจำเป็นของระบบควบคุมแบบปิดก็คือส่วนป้อนกลับ ซึ่งหมายถึงสัญญาณที่ได้มาจากค่าตัวแปรที่ต้องการจริงๆ เปลี่ยนเป็นสัญญาณแล้วป้อนกลับ เพื่อเปรียบเทียบกับค่าของตัวแปรที่ต้องการ การป้อนกลับนี้จะถือว่าเป็นการป้อนกลับแบบลบ (Negative Feedback) เมื่อสัญญาณป้อนกลับนี้นำไปลบออกจากค่าที่ต้องการหรือค่ามาตรฐาน นั่นคือ

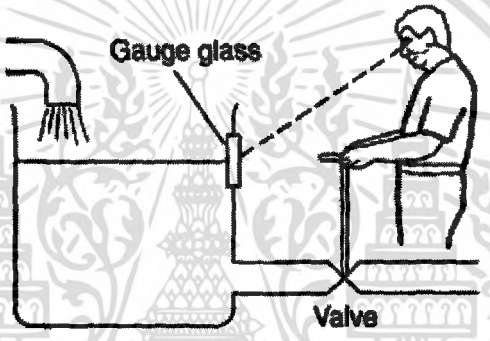
ความผิดพลาด = ค่าสัญญาณอ้างอิง – ค่าสัญญาณป้อนกลับ (1)

การป้อนกลับแบบลบนี้ มีความจำเป็นในการที่เราต้องการให้ค่าตัวแปรที่เราต้องการควบคุมมีค่าตรงกับความต้องการของเราคือค่าของสัญญาณมาตราฐาน ส่วนการป้อนกลับแบบบวก (Positive Feedback) นั้น จะเกิดขึ้นเมื่อสัญญาณป้อนกลับจะนำมารวมกับค่ามาตรฐาน นั่นคือ

ความผิดพลาด = ค่าสัญญาณอ้างอิง + ค่าสัญญาณป้อนกลับ (2)

ในรูปที่ 2.1 นั้นเราจะเห็นว่าสัญญาณป้อนกลับจะนำเข้ามารวมกับสัญญาณมาตรฐานที่ส่วนเปรียบเทียบ (Comparison Element) โดยส่วนเปรียบเทียบ (Comparison Element) นี้นิยมจะเขียน

เป็นวงกลม ซึ่งมีกากบาทอยู่ตรงกลาง ซึ่งเป็นสัญลักษณ์มาตรฐานของการบวกสัญญาณ สำหรับในส่วนของ ส่วนเปรียบเทียบ (Comparison Element) ของระบบป้อนกลับแบบลบ (Negative Feedback) เราจะเห็นว่าค่าสัญญาณมาตรฐาน จะถูกกำหนดด้วยเครื่องหมายบวก (+) และค่าสัญญาณป้อนกลับจะกำหนดด้วยเครื่องหมายลบ (-) ดังนั้นสัญญาณที่ออกมาจะเป็นความแตกต่างระหว่างสัญญาณทั้งสอง ถ้าหากว่าการป้อนกลับเป็นการป้อนกลับแบบบวก เครื่องหมายกำหนดสัญญาณที่ส่วนเปรียบเทียบ (Comparison Element) ก็จะมีเครื่องหมายเป็นบวก (+) ด้วย เพื่อแสดงถึงส่วนประกอบต่างๆ ของระบบควบคุมแบบปิด เราลองพิจารณาระบบควบคุมการใส่น้ำเข้าถังอีกครั้งหนึ่ง ดังแสดงในรูปที่ 2.2 โดยในครั้งนี้ผู้ควบคุมการเปิดปิดน้ำ สามารถมองเห็นระดับน้ำในถังได้โดยการมองผ่านเกจกลาส (Gauge Glass)

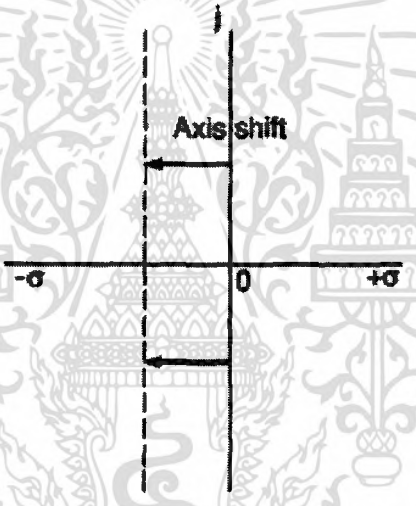


รูปที่ 2.2 ระบบควบคุมการเปิดน้ำเข้าสู่ถังแบบปิด

2.2 ความเสถียรภาพของการควบคุมแบบป้อนกลับ

การใช้ข้อกำหนดการเสถียรของระบบโดยใช้วิธีการเร้า-เฮอร์วิธ (Routh-Hurwitz Criterion) จะให้คำตอบของความเสถียรได้ในส่วนหนึ่ง แต่อย่างไรก็ตามยังมีส่วนสำคัญอีกส่วนหนึ่งในการพิจารณาความเสถียรของระบบ ซึ่งก็คือ การวิเคราะห์ระบบที่เสถียรนั้นว่ามีความเสถียรมากน้อยเพียงใด หรือเรียกอีกกรณีหนึ่งว่า ความเสถียรสัมพันธ์ (Relative Stability)

พิจารณาความเสถียรสัมพันธ์ของระบบ เพื่อที่จะดูว่าระบบที่เสถียรของเรานั้นอยู่ห่างจากความไม่เสถียรมากน้อยเท่าใด ในการที่จะหาค่านี้จำเป็นอย่างยิ่งที่จะต้องทราบว่ารากของเรานั้นอยู่ใกล้จุด 0 มากน้อยเพียงใด เราสามารถทำเช่นนี้ได้โดยขยับแกน $j\omega$ ไปทางซ้ายมือเป็นจำนวนจำนวนหนึ่ง และคำนวนหาว่าเราจะต้องเลื่อนแกนไปมากน้อยเพียงใดเพื่อระบบจะไม่เสถียร ลักษณะการขยับแกนแสดงในรูปที่ 2.3



รูปที่ 2.3 การขยับแกน

การขยับแกนไปเป็นขนาด $-\sigma$ หมายถึง เทอมเศษของฟังก์ชันถ่ายโอนหรือสมการเฉพาะนั้นจะแทนค่า s ด้วย $(r-\sigma)$ โดย r เป็นปริมาณใหม่ที่เราจะใช้ตรวจสอบความเสถียร ตัวอย่างเช่น พิจารณาสมการเฉพาะ

$$q(s) = s^3 + 4s^2 + 8s + 4$$

ซึ่งเราได้ Routh Array เป็น

$$\begin{array}{c|cc} s^3 & 1 & 8 \\ s^2 & 4 & 4 \\ s^1 & 2 & 0 \\ s^0 & 4 & \end{array}$$

ซึ่งระบบเป็นระบบเสถียร ถ้าหากเราขยับแกนไปที่ -1 เราจะแทน s ด้วย (r-1) ซึ่งทำให้เราได้สมการเฉพาะในระบบแกนใหม่เป็น

$$q(r) = (r-1)^3 + 4(r-1) + 8(r-1) + 4$$

นั่นคือ

$$q(r) = (r^3 - 3r^2 + 3r - 1) + 4(r^2 - 2r + 1) + 8(r-1) + 4$$

หรือเมื่อจัดรูปเราจะได้

$$q(r) = r^3 + r^2 + 3r - 1$$

เราจะได้ตารางของเร้า(Routh Array) เช่น

$$\begin{array}{cccc} r^3 & 1 & 3 \\ r^2 & 1 & -1 \\ r^1 & 4 & \\ r^0 & -1 & \end{array}$$

ซึ่งเราจะพบว่าระบบนี้ไม่เสถียร และเนื่องจากมีการเปลี่ยนเครื่องหมายเพียงหนึ่งครั้งเราจึงได้ว่าจะมีค่ารากอยู่ค่าหนึ่ง ที่อยู่ทางขวามือของเส้น –1

2.3 รูปแบบการควบคุม

คอนโทลเลอร์ (Controller) เป็นส่วนประกอบหนึ่งในระบบควบคุมแบบป้อนกลับ ซึ่งจะมีสัญญาณความผิดพลาดเป็นอินพุทและมีเอาท์พุทเป็นอินพุทของ (Corrective Element) บทนี้จะกล่าวถึงวิธีการเลือกใช้คอนโทลเลอร์ที่เหมาะสมกับระบบควบคุมแบบป้อนกลับและวิธีการหาพารามิเตอร์ที่เหมาะสมสำหรับคอนโทลเลอร์ โดยที่ ความสัมพันธ์ระหว่างอินพุทและเอาท์พุทของคอนโทลเลอร์ เรานิยมเรียก กฎการควบคุม (Control Law) โดยจะมี 3 รูปแบบ คือ

1. Proportional Control
2. Integral Control
3. Derivative Control

ในระบบบางประเภท เรามีความจำเป็นที่ต้องปรับปรุงสมรรถนะของระบบควบคุม ซึ่งสามารถทำได้โดยการใช้ส่วนประกอบอื่นเพิ่มเติมเข้ากับระบบควบคุม การที่เราปรับเปลี่ยนสมรรถนะของระบบควบคุมป้อนกลับแบบนี้เราเรียกว่า การชดเชย (Compensation)

ระบบควบคุมแบบสัดส่วน (Proportional Control)

ในระบบควบคุมแบบสัดส่วน เราจะได้ว่า เอาท์พุทของคอนโทลเลอร์จะเป็นสัดส่วนกับอินพุทของคอนโทลเลอร์ และถ้าเรากำหนดสัญญาณอินพุทที่ให้กับคอนโทลเลอร์เป็น ค่าความผิดพลาด(e) ซึ่งเป็นฟังก์ชันของเวลา เราจะได้สมการของเอาท์พุทเป็น

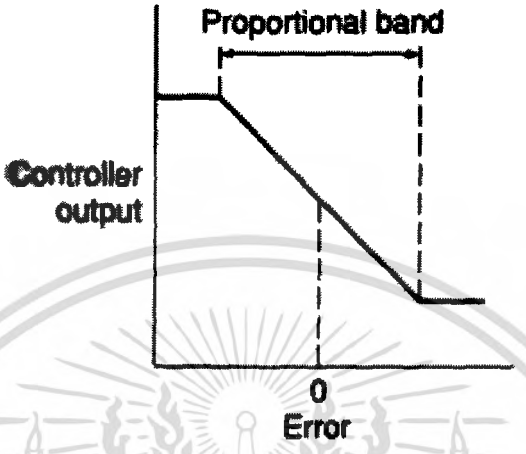
$$Output = K_p e \qquad 2.3.1$$

เมื่อ K_p เป็นค่าคงที่เรียก พร็อพเพอร์ชันนอลเกน (Proportional Gain) เราจะพบว่าเอาท์พุทที่ออกจากคอนโทลเลอร์แบบสัดส่วน (Proportional Control) จะขึ้นกับขนาดของความผิดพลาดในขณะที่เรากำลังพิจารณา ทำให้ฟังก์ชันถ่ายโอนของคอนโทลเลอร์ $G_c(s)$ จะมีค่าเป็น

$$G_c(s) = K_p \qquad 2.3.2$$

ดังนั้นการควบคุมด้วยคอนโทลเลอร์แบบนี้ก็จะเป็นเพียงการขยายสัญญาณความผิดพลาดเท่านั้น การที่เราได้สัญญาณความผิดพลาดขนาดใหญ่ที่เวลาหนึ่ง จะทำให้เกิดเอาท์พุทที่มีขนาดใหญ่จากคอนโทลเลอร์ในเวลานั้น อย่างไรก็ตามการที่เรากำหนดให้เกน (Gain) คงที่นั้นในทางปฏิบัติเราอาจจะกำหนดไว้ในบางช่วงของสัญญาณความผิดพลาดเท่านั้น เราอาจกำหนดให้คอนโทลเลอร์ของเรามีค่าเอาท์พุทไม่น้อยกว่าค่าค่าหนึ่งและไม่มากเกินกว่าค่าค่าหนึ่งก็ได้ ซึ่งการ

กำหนดช่วงจำกัดของเอาท์พุทจะมีลักษณะดังรูปที่ 2.4 และการกำหนดเอาท์พุทแบบระบบควบคุมสัดส่วน (Proportional Control) ช่วงที่มีการกำหนดสัดส่วนนี้ เราจะเรียกว่าพร็อพเพอร์ชันนอลแบนด์(Proportional band.)

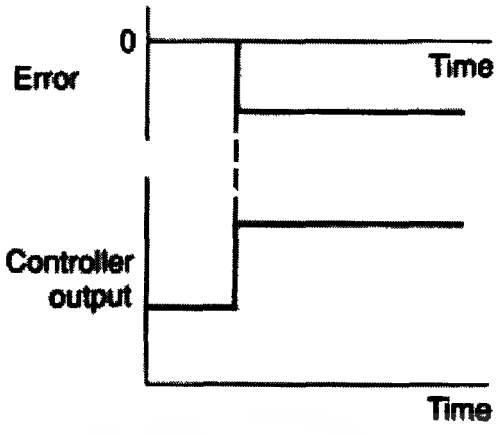


รูปที่ 2.4 การกำหนดช่วงจำกัดของเอาท์พุท

การกำหนดพร็อพเพอร์ชันนอลแบนด์ (Proportional band.)นี้ จะช่วยให้สัญญาณเอาท์พุทมีค่าจำกัดไม่ไปสู่ค่าอนันต์ ทั้งทางด้านบวก (+) และทางด้านลบ (-) และเมื่อคอนโทลเลอร์มีเอาท์พุทสูงที่สุดที่เป็นไปได้ค่าหนึ่งแล้ว เราก็นิยมที่จะกำหนดเอาท์พุทค่าใด ๆ เป็นร้อยละของค่าสูงสุดที่เป็นไปได้ ดังนั้นการเปลี่ยนแปลงค่าเอาท์พุทของคอนโทลเลอร์ 100% ก็หมายถึงว่าเอาท์พุทจะเปลี่ยนจากค่าต่ำสุดที่เป็นไปได้ ไปเป็นค่าสูงสุดที่เป็นไปได้ ซึ่งจะทำให้เราได้ว่า

$$K_p = \frac{100}{proportionalband} \qquad 2.3.3$$

เนื่องจากเอาท์พุทของคอนโทลเลอร์จะเป็นสัดส่วนกับอินพุทดังนั้น ถ้าหากอินพุทมีลักษณะเป็น สัญญาณหนึ่งหน่วย (Unit step) เอาท์พุทที่ได้ก็จะมีลักษณะเป็นสัญญาณหนึ่งหน่วย (Unit step) เช่นกัน โดยลักษณะของกราฟแสดงอินพุทและเอาท์พุทจะมีสัดส่วนที่แน่นอนค่าหนึ่ง ตามรูปที่ 2.5 โดยรูปนี้แสดงถึงการตอบสนองของคอนโทลเลอร์เมื่ออินพุทอยู่ในช่วงพร็อพเพอร์ชันนอลแบนด์ (Proportional band.)

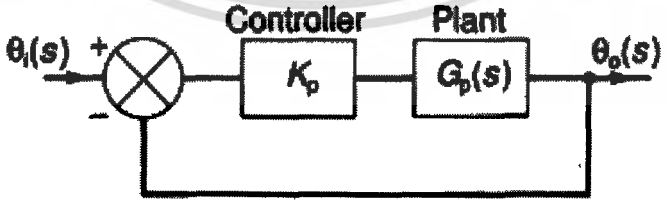


รูปที่ 2.5 การตอบสนองของคอนโทลเลอร์แบบ Proportional Control

ในทางปฏิบัติ ระบบควบคุมแบบสัดส่วน (Proportional Control) นี้จะมีลักษณะเหมือนกับเครื่องขยายสัญญาณรูปแบบหนึ่ง ซึ่งอาจจะเป็นในลักษณะของอุปกรณ์ไฟฟ้า หรืออาจจะเป็นเครื่องขยายสัญญาณเชิงกล เช่น คาน ก็ได้ ลักษณะของระบบที่ควบคุมแบบสัดส่วน (Proportional Control) จะมีลักษณะดังที่แสดงในรูปที่ 2.6 และจะทำให้ได้ฟังก์ชันถ่ายโอนระบบเปิดเป็น

$$G_0(s) = K_p G_p(s) \qquad 2.3.4$$

เมื่อ $G_p(s)$ เป็นฟังก์ชันถ่ายโอนของระบบ



รูปที่ 2.6 ระบบควบคุมแบบ Proportional control

ข้อเสียประการสำคัญของระบบควบคุมที่คอนโทลเลอร์คือ ไม่ได้มีการเพิ่มเทอม $\frac{1}{s}$ (หรือการเพิ่มปริพันธ์) ในส่วนฟอร์เวิร์ดพาธ (Forward Path) ซึ่งหมายความว่า ถ้าระบบเป็นระบบรูปแบบศูนย์ (type 0) คอนโทลเลอร์จะไม่ได้เปลี่ยนแปลงรูปแบบ (type) ของระบบทำให้ระบบเป็นรูปแบบศูนย์ (type0) เหมือนเดิม และทำให้เกิดความผิดพลาดที่สภาพคงตัว เนื่องจากคอนโทลเลอร์ไม่ได้ทำการเพิ่มโพล(Pole)หรือซีโร่(Zero)ใหม่ให้กับระบบเพียงแต่เปลี่ยนตำแหน่งของโพล(Pole)หรือซีโร่(Zero)เท่านั้นเนื่องจากระบบควบคุมแบบป้อนกลับหนึ่งหน่วย ตามรูปที่ 2.6 จะมีฟังก์ชันถ่ายโอนของระบบเป็น

$$G(s) = \frac{K_p G_p(s)}{1 + K_p G_p(s)} \qquad 2.3.5$$

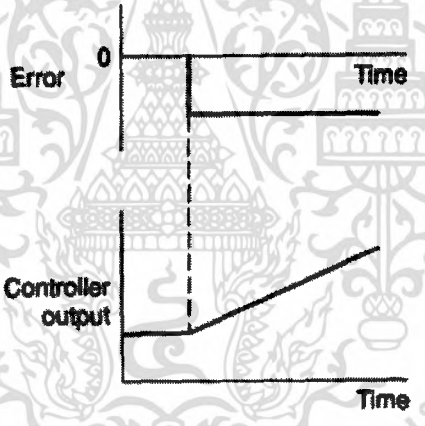
และสมการคุณลักษณะจะเป็น $(1 + K_p G_p(s))$ และมีรากเปลี่ยนไปตามค่าของ K_p

ระบบควบคุมแบบปริพันธ์ (Integral Control)

ในระบบควบคุมแบบปริพันธ์เอาท์พุทของคอนโทลเลอร์จะเป็นสัดส่วนกับปริพันธ์ของสัญญาณผิดพลาดเทียบกับเวลา หรือ

$$Output = K_I \int_0^t edt \qquad 2.3.6$$

เมื่อ K_p เป็นค่าคงที่เรียกว่าอินทิกรัลเกน (Integral Gain) ซึ่งจะมีหน่วยเป็น 1/sec รูปที่ 2.7 แสดงลักษณะการตอบสนองของระบบควบคุมแบบปริพันธ์ (Integral Control) เมื่อได้รับสัญญาณความผิดพลาดแบบสัญญาณหนึ่งหน่วย (Unit step) ค่าปริพันธ์ระหว่างเวลา t และ 0 จะหมายถึงพื้นที่ใต้กราฟของสัญญาณความผิดพลาดจากเวลา 0 ถึง t ดังนั้นเนื่องจากเริ่มการมีสัญญาณความผิดพลาดแบบสัญญาณหนึ่งหน่วย (Unit step) เอาท์พุทที่ออกจากคอลโทลเลอร์จะมีค่ามากขึ้นเรื่อย ๆ ด้วยอัตราที่คงที่ ทำให้เอาท์พุทที่เวลาใดๆจะเป็นสัดส่วนกับความผิดพลาดที่เกิดขึ้น

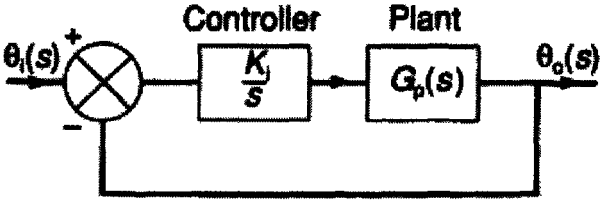


รูปที่ 2.7 ลักษณะการตอบสนองของคอนโทลเลอร์แบบ Integral Control

เปลี่ยนรูปลาปลาสของสมการ 2.3.6 จะทำให้เราได้ฟังก์ชันถ่ายโอนของคอนโทลเลอร์เป็น

$$G_c(s) = \frac{Output(s)}{e(s)} = \frac{K_I}{s} \qquad 2.3.7$$

ดังนั้นสำหรับระบบที่แสดงในรูปที่ 2.8 การควบคุมแบบปริพันธ์ (Integral Control) จะให้ forward-path transfer function เป็น $\frac{K_I}{s} G_p(s)$ และทำให้มีฟังก์ชันถ่ายโอนระบบเปิดเป็น



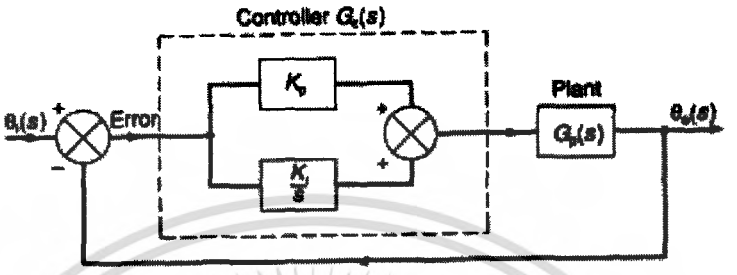
รูปที่ 2.8 ระบบควบคุมแบบ Integral Control

เราสามารถพิจารณาถึงข้อได้เปรียบของการควบคุมแบบปริพันธ์ (Integral Control) ได้จากสมการ 2.3.7 ซึ่งเราจะเห็นว่าระบบควบคุมแบบปริพันธ์ (Integral Control) จะเพิ่มจำนวนโพล(Pole)ให้กับระบบควบคุมและเพิ่ม type ของระบบ จาก type 0 เป็น type 1 ซึ่งทำให้ระบบมีความผิดพลาดที่สภาวะคงตัวเป็นศูนย์เทียบต่อ step input อย่างไรก็ตามการเพิ่มโพลที่ S = 0 และไม่มีการเพิ่มซีโร่ให้กับระบบควบคุมจะทำให้ความแตกต่างระหว่างจำนวนโพล (n) และจำนวนซีโร่ (m) เพิ่มขึ้นอีก 1 ซึ่งจะมีผลให้ asymptote angles ของทางเดินรากลดลง และจุดตัดจะเคลื่อนไปทางครึ่งขวาของ s-plane มากขึ้น มีผลทำให้ความเสถียรสัมพัทธ์ของระบบลดลง

$$\text{Asymptote angle } = \pm\frac{\pi}{n-m}, \frac{3\pi}{n-m}, \ldots\ldots$$

การควบคุมแบบสัดส่วนร่วมกับปริพันธ์ (Proportional Plus Integral Control)

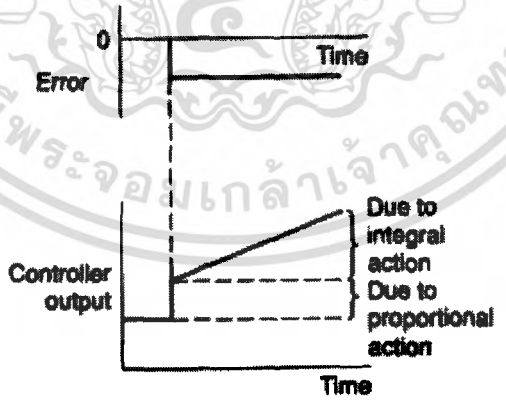
การที่ระบบควบคุมมีความเสถียรสัมพัทธ์ลดลง เมื่อเราใช้การควบคุมแบบปริพันธ์สามารถที่จะแก้ไขได้ในระดับหนึ่งโดยการใช้การควบคุมแบบสัดส่วนร่วมกับแบบปริพันธ์ (Proportional Plus Integral, PI) ซึ่งลักษณะของระบบควบคุมจะเป็นตามรูปที่ 2.9



รูปที่ 2.9 แผนภาพบล็อกที่ประกอบด้วย Proportional Plus Integral

สำหรับระบบดังกล่าวจะมีเอาท์พุทของคอนโทลเลอร์เป็น

$$Output = K_p e + K_I \int_0^t e dt \qquad 2.3.8$$



รูปที่ 2.10 การตอบสนองของ PI Controller

รูปที่ 2.10 แสดงเอาท์พุทของคอนโทลเลอร์ที่ได้รับเมื่อมีอินพุทเป็นสัญญาณความผิดพลาดแบบ step ถ้าเราเปลี่ยนรูปลาปลาส ของสมการ 2.3.7 เราจะได้ฟังก์ชันถ่ายโอนของคอนโทลเลอร์แบบ PI เป็น

$$G_0(s) = K_p + \frac{Ki}{s}$$

$$= \frac{sK_p + Ki}{s}$$

$$= K_p \frac{(s + Ki/K_p)}{s}$$

เราให้ Integral Time Constant τ_i เป็น

$$\tau_i = \frac{K_p}{Ki}$$

ดังนั้นเราจะได้

$$G_c(s) = K_p \frac{(s + 1/\tau_i)}{s} \qquad 2.3.9$$

และจะทำให้เราได้ฟังก์ชันถ่ายโอนระบบเปิดเป็น

$$G_0(s) = G_c(s) G_p(s)$$

$$= \frac{K_p[s + (\frac{1}{\tau_i}) G_p(s)]}{s}$$

เราจะเห็นว่าเรามีซีโร่ที่ $s = -\frac{1}{\tau_i}$ และโพลที่ S=0 เพิ่มให้กับฟังก์ชันถ่ายโอนของระบบเมื่อเราใช้การควบคุมแบบ PI การที่เราเพิ่มตัวประกอบ S เข้ากับเทอมส่วนของฟังก์ชันถ่ายโอนก็เสมือนกับเราเพิ่มแบบของระบบขึ้นไป 1 จึงทำให้ระบบนี้จะไม่มีความผิดพลาดที่สภาพคงตัวสำหรับอินพุทแบบขั้นบันได นอกจากนั้นการที่เราเพิ่มซีโร่ให้กับระบบไปพร้อมๆกันก็จะทำให้ความ

แตกต่างระหว่างจำนวนโพล n และจำนวนซีโร่ m มีค่าคงที่ ดังนั้นมุมของ asymptote สำหรับทางเดินของรากมีค่าคงเดิม

อย่างไรก็ตามจุดตัดของเส้น asymptotes บนแกนจริง จะเคลื่อนที่เข้าหาจุดกำเนิดมากขึ้น ยังผลให้ความเสถียรของระบบลดลงบ้าง

Intersection/point=(ผลรวมของโพล-ผลรวมของซีโร่)/(n-m)

การเพิ่มโพลที่ S=0 และซีโร่ ที่ $s=-\frac{1}{\tau_i}$ จะทำให้จุดต้องเปลี่ยนไปเท่ากับ $\pm\frac{(1/\tau_i)}{(n-m)}$ ซึ่งจะทำให้มีค่าเป็นบวกมากขึ้น และจุดตัดจะเคลื่อนที่มาทางขวามือเข้าใกล้จุดกำเนิดมากขึ้น อย่างไรก็ตาม การลดลงของความเสถียรสัมพัทธ์นี้จะน้อยกว่าการที่เราใช้การควบคุมแบบปริพันธ์เพียงอย่างเดียว

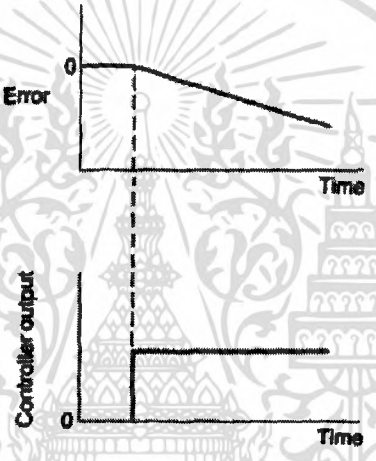
ค่าของ K_p และ K_i จะเป็นค่าที่ใช้กำหนดตำแหน่งของซีโร่(Zero)และโพล(Pole)ของระบบ โดยตำแหน่งของซีโร่จะกำหนดด้วยค่า K_p ในขณะที่ K_i จะเป็นค่าที่ใช้กำหนดโพลระบบปิด

การควบคุมแบบอนุพันธ์ (Derivative Control)

การควบคุมอีกแบบหนึ่งก็คือการควบคุมแบบอนุพันธ์ (Derivative Control) การควบคุมแบบนี้เอาท์พุทจะเป็นสัดส่วนกับอัตราการเปลี่ยนแปลงความผิดพลาดเทียบต่อเวลา นั่นคือ

$$Output = K_d \frac{de}{dt} \qquad 2.3.10$$

เมื่อ K_d คือ Derivative Gain และมีหน่วยเป็นวินาที



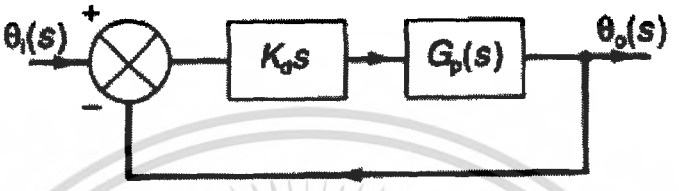
รูปที่ 2.11 การตอบสนองของ Derivative Control

รูปที่ 2.11 แสดงสิ่งที่เกิดขึ้นเมื่อสัญญาณความผิดพลาดเป็นสัญญาณแบบ Ramp เมื่อเริ่มได้รับสัญญาณความผิดพลาดและไม่ใช่ค่าของความผิดพลาดซึ่งทำให้เราได้สัญญาณส่งออกจากคอนโทลเลอร์มีค่ามากก่อนที่จะเกิดความผิดพลาดขึ้นมากจริง ๆ อย่างไรก็ตามหากความผิดพลาดมีค่าคงที่ก็จะไม่มีการสะสมค่าความผิดพลาดแม้ว่าค่าความผิดพลาดจะมีมากก็ตาม ทำให้การควบคุมแบบอนุพันธ์นี้ไม่อ่อนไหวต่อค่าความผิดพลาดที่คงที่หรือเปลี่ยนแปลงอย่างช้า ๆ ซึ่งผลที่ตามมาการควบคุมแบบนี้จะไม่ใช้เพียงตัวเดียวแต่มักจะใช้ควบคุมร่วมกับการควบคุมแบบอื่นเปลี่ยนรูปลาปลาสสมการ 2.3.10 เพื่อที่จะหาฟังก์ชันถ่ายโอนของคอนโทลเลอร์ ซึ่งจะเป็น

$$G_c(s) = K_d s \qquad 2.3.11$$

ดังนั้นสำหรับระบบควบคุม ดังที่แสดงในรูปที่ 2.12 การที่มีการควบคุมแบบอนุพันธ์จะทำให้เราได้ฟังก์ชันถ่ายโอนเป็น

$$G_0(s) = \frac{K_d s G_p(s)}{1 + K_d s G_p(s)} \qquad 2.3.12$$



รูปที่ 2.12 แผนภาพบล็อกที่ประกอบด้วย Derivative Control

ถ้าหากว่าระบบเป็นแบบ type 1 หรือสูงกว่า การควบคุมแบบอนุพันธ์จะลด S ในเทอมส่วนลง และลด type ของระบบลง 1 อย่างไรก็ตามเราได้กล่าวก่อนหน้านี้แล้วว่า การควบคุมแบบอนุพันธ์นี้มักจะไม่ใช้เพียงลำพังแต่เราจะใช้ร่วมกับการควบคุมแบบอื่น เพราะเมื่อเราใช้การควบคุมแบบอนุพันธ์จะทำให้เราเพิ่มความเร็วในการตอบสนองของระบบต่อความผิดพลาดที่เกิดขึ้น

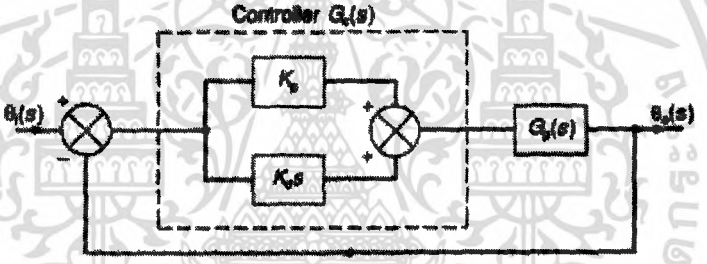
ในทางปฏิบัติการนำกฎการควบคุมแบบอนุพันธ์ไปใช้นั้นค่อนข้างจะลำบาก ดังนั้นในทางปฏิบัติโดยทั่วไปจะเป็นการประมาณการควบคุมแบบอนุพันธ์ โดยใช้ Lead Compensator

การควบคุมแบบสัดส่วนร่วมกับอนุพันธ์ (Proportional Plus Derivative Control)

ถ้าการควบคุมแบบอนุพันธ์ใช้ร่วมกับการควบคุมแบบสัดส่วน (PD) ดังที่แสดงในรูปที่ 2.12 เราจะได้ฟังก์ชันถ่ายโอนระบบเปิดเป็น

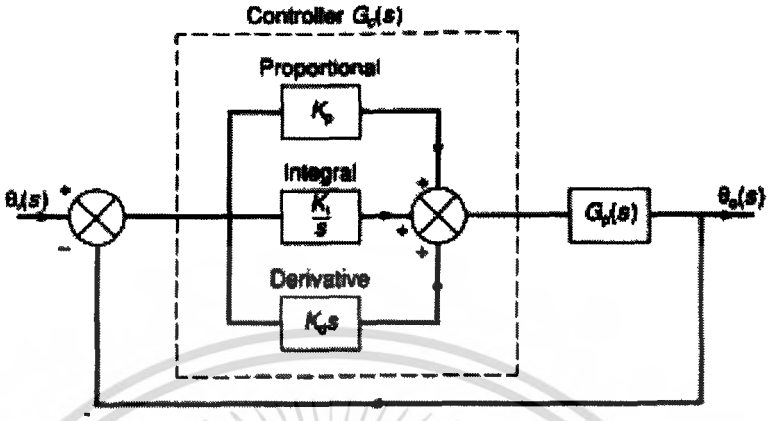
$$G_0(s) = (K_p + K_d s)G_p(s)$$
$$G_0(s) = K_d[(1/\tau_i) + s]G_p(s) \qquad 2.3.13$$

เมื่อ $\tau_d = \frac{K_p}{K_d}$ คือ Derivative Time Constant ซึ่งในการควบคุมแบบนี้จะมีซีโร่เพิ่มขึ้นที่ $s = -\frac{1}{\tau_d}$ และจะเห็นว่าไม่มีการเปลี่ยนแปลงของระบบ ทำให้ไม่มีการเปลี่ยนแปลงค่าความผิดพลาดที่สภาวะคงตัว



รูปที่ 2.13 แผนภาพบล็อกที่ประกอบด้วย PD Control

PID control



รูปที่ 2.14 Block Diagram ที่ประกอบด้วย PID control

การควบคุมโดยใช้แบบสัดส่วนรวมกับแบบปริพันธ์และรวมกับแบบอนุพันธ์ (PID control) หรือ ที่เรียก การควบคุมแบบ 3 เทอม (tree-term control) ระบบจะมีลักษณะตามรูปที่ 2.14 จะทำให้เอาท์พุทของคอนโทลเลอร์เมื่อรับอินพุทเป็นความผิดพลาด e ดังนี้

$$Output = K_p e + K_i \int_0^t e dt + K_d \frac{de}{dt} \qquad 2.3.14$$

ฟังก์ชันถ่ายโอนของคอนโทลเลอร์ จะเป็น

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s \qquad 2.3.15$$

ซึ่งเราสามารถจัดรูปได้เป็น

$$G_c(s) = K_p[1 + \frac{K_i}{K_p s} + \frac{K_d s}{K_p}]$$

หรือ $$G_c(s) = K_p(1 + \frac{1}{\tau_i s} + \tau_d s) \qquad 2.3.16$$

ฟังก์ชันถ่ายโอนระบบเปิดของระบบที่แสดงในรูปที่ 6.12 จะเป็น

$$G_0(s) = G_c(s)G_p(s) = K_p(1+\frac{1}{\tau_i s}+\tau_d s)G_p(s)$$

$$G_0(s) = \frac{K_p(\tau_i s+1+\tau_i\tau_d s^2)G_p(s)}{\tau_i s} \qquad 2.3.17$$

ดังนั้นการควบคุมแบบ PID controller จะเพิ่มจำนวนซีโร่ให้กับระบบเท่ากับ 2 และเพิ่มจำนวนโพล 1 โพล และทำให้ชนิด type ระบบเพิ่มขึ้น

บทที่ 3
หลักการและโครงสร้างการออกแบบ

3.1 หลักการออกแบบ

การออกแบบโปรแกรม JKI Toolbox นี้ มีการออกแบบโปรแกรมโดยศึกษาข้อมูลเพื่อทำการออกแบบ ประกอบด้วย

- การศึกษาการทำงานของโปรแกรม MATLAB สำหรับการคำนวณทางด้านระบบควบคุม และศึกษาถึงรูปแบบฟังก์ชันการใช้งานทางด้านต่างๆ ที่เกี่ยวข้องกับระบบควบคุมของโปรแกรม MATLAB เช่น ฟังก์ชันการหาโพล ซีโร่ ฟังก์ชันการคำนวณหา Root Locus ฟังก์ชันการแปลง Transfer Function เป็น State space
- ศึกษาการทำงานของเครื่องคำนวณ (Calculator) รุ่น TI-89 และศึกษาความสามารถ รวมถึงคุณสมบัติของตัวเครื่องในการนำไปใช้งาน ในการคำนวณทางด้านคณิตศาสตร์ และศึกษาความสามารถของหน่วยความจำของเครื่องคำนวณ ความสามารถในการประมวลผลของหน่วยประมวลกลาง (CPU) ของเครื่องคำนวณ สำหรับการรองรับโปรแกรมที่มีความซับซ้อนมาทำงานในตัวเครื่อง
- ศึกษาการทำงานของโปรแกรมที่ทำงานในเครื่องคิดเลขที่มีลักษณะคล้ายคลึงกัน และศึกษาถึงวิธีการเขียนโปรแกรมที่สามารถใช้งานในเครื่องคิดเลข
- การออกแบบโปรแกรม JKI Toolbox สร้างขึ้นจากการเรียนแบบตัวโปรแกรม MATLAB ที่ใช้งานภายในเครื่องคอมพิวเตอร์
- วิธีที่ใช้ออกแบบโปรแกรมและสร้างโปรแกรม JKI Toolbox เป็นการวิจัยทดลองโดยเทียบกับ
  โปรแกรมMATLAB

3.2 เครื่องมือในการออกแบบโปรแกรมJKI Toolbox

- Personal Computer

  -Pentium 4 2.44 GHz

  -RAM 1 GB

  -Hardisk 80 GB

  -LCD 17inch

- LABTOP HP

  -Pentium 4 2.0 GHz

  -RAM 1 GB

  -Hardisk 80 GB

  -LCD 17inch

- MATLAB Program
- Ti-89 Program
- Ti-89 Calculator

รูปที่ 3.2.1เครื่องคิดเลขรุ่นTi-89

3.3 โครงสร้างของฟังก์ชันที่ออกแบบและรายละเอียด

จากการศึกษาและทดลองการใช้โปรแกรมJKI Toolbox สามารถออกแบบสร้างฟังก์ชันการใช้งานของโปรแกรมได้ดังนี้

Function ABOUT()

```
ABOUT()
Prgm
Dialog
 Title "A B O U T  U S"
 Text ""
 Text "Natthapong 47010377"
 Text "Khemjira 47010789"
 Text "Itthinan 47010993"
 Text "KMITL2008"
 Text ""
 Text "Special Thanks to Dr.Poramate"
 Text "Dr.Poramate& Brookie"
 Text ""
EndDlog
EndPrgm
```

Function QUIT()

```
QUIT()
Prgm
CustmOff
EndPrgm
```

Function JKI()

```
JKI()
Prgm
Custom
Title "Files"
Item "jki\About()"
Item "Quit()"
Title "Models"
Item "jki\ss("&char(2)&",,,)"
Item "jki\tf("&char(2)&",)"
Item "jki\zpk("&char(2)&",,)"
Item "jki\clloop("&char(2)&",)"
Title "Data"
Item "jki\gettd("&char(2)&")"
Item "jki\tf2ss("&char(2)&")"
Item "jki\zpkdata("&char(2)&")"
Title "Dynamics"
Item "jki\dcgain("&char(2)&")"
Item "jki\cPoles("&char(2)&",)"
Item "jki\pzmap("&char(2)&")"
Title "Analysis"
Item "jki\step("&char(2)&")"
Item "jki\rlocus("&char(2)&",{})"
Item "jki\PID("&char(2)&")"
Title "Tools"
Item "jki\logspace("&char(2)&",,)"
Item "jki\mag("&char(2)&")"
Item "jki\phase("&char(2)&")"
Item "jki\poly2cof("&char(2)&",)"
Item "jki\roots("&char(2)&")"
Item "jki\routh("&char(2)&",)"
Item "jki\ZoomFit2()"
```

```
Item  "jki\Invlap("&char(2)&")"
EndCustm
CustmOn
EndPrgm
```

Function SS(a,b,c,d)

```
SS(a,b,c,d)
Func
Local o
If getType(a)="MAT" Then
c*(s*identity(rowDim(a))-a)^(-1)*b⟶o
If when(d≠0,true,false,false)
 o+d⟶o
If dim(o)={1,1}
 o[1,1]⟶o
Else
 c/(s-a)*b+d⟶o
EndIf
comDenom(o)
EndFunc
```

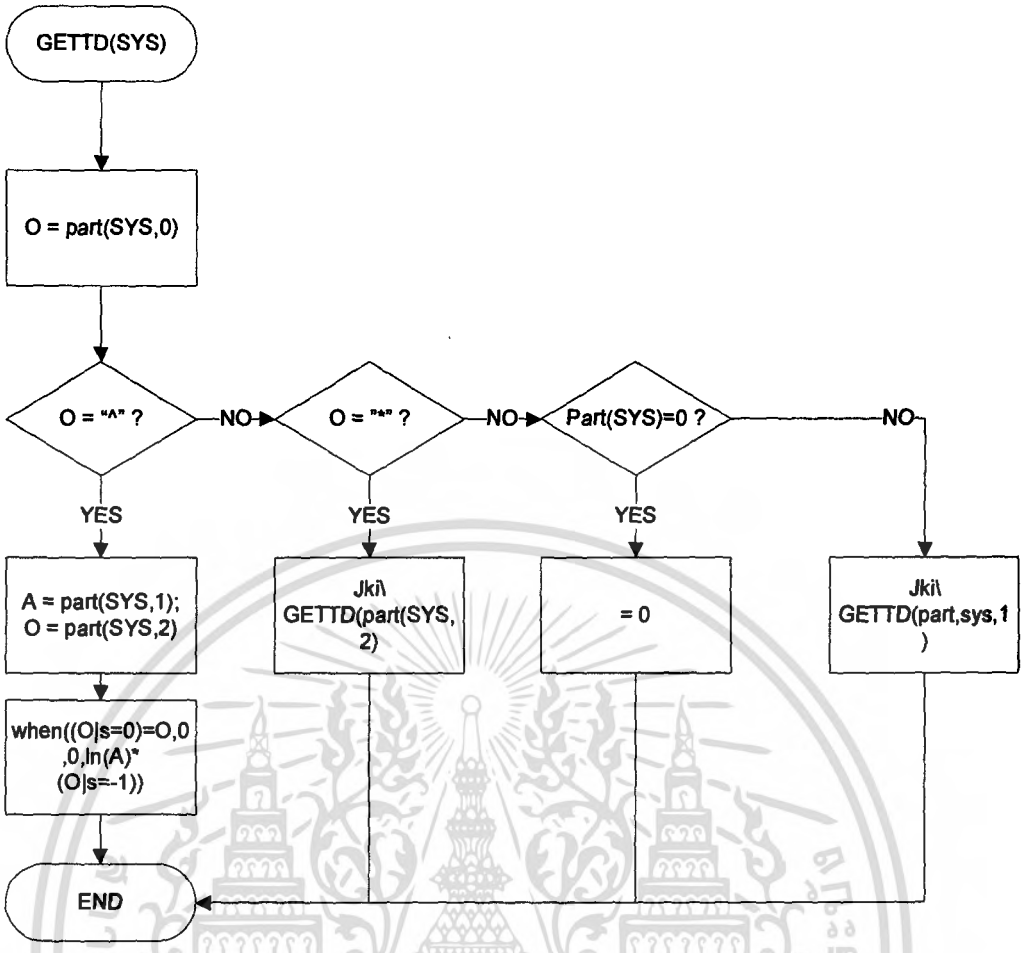
Function TF(n,d)

```
TF(n,d)
Func
Local t
getType(n)⟶t
If t="NUM" or t="EXPR" or t="VAR"
 {n}⟶n
getType(d)⟶t
If t="NUM" or t="EXPR" or t="VAR"
 {d}⟶d
If getType(n)≠"LIST" or getType(d)≠"LIST"
 Return "Argument error"
polyEval(n,s)/(polyEval(d,s))
EndFunc
```

Function ZPK(z,p,k)

```
ZPK(z,p,k)
Func
Local t
getType(z)⟶t
If t="NUM" or t="EXPR" or t="VAR":{z}⟶z
getType(p)⟶t
If t="NUM" or t="EXPR" or t="VAR":{p}⟶p
If getType(p)"LIST" or getType(z)"LIST":Return "Argument error"
k*Π(s-z[i],i,1,dim(z))/(Π(s-p[i],i,1,dim(p)))
EndFunc
```

```
Function GETTD(sys)
GETTD(sys)
Func
Local a,o
part(sys,0) ⟶o
If o="^" Then
part(sys,1)⟶a
part(sys,2)⟶o
when((o|s=0)=o,0,0,ln(a)*(o|s=-1))
ElseIf o="*" Then
jki\gettd(part(sys,2))
ElseIf part(sys)=0 Then
0
Else
jki\gettd(part(sys,1))
EndIf
EndFunc
```



รูปที่ 3.3.1 Flow Chart ของ ฟังก์ชันGETTD

```
Function TF2SS(f)
TF2SS(f)
Prgm
Local x,y,z,t
jki\tf2ss2(f)⟶f
expr(f[1])⟶x
expr(f[2])⟶y
expr(f[3])⟶z
expr(f[4])⟶t
Disp "A="
Pause x
Disp "B="
Pause y
Disp "C="
Pause z
Disp "D="
Pause t
Dialog
Text "Save matrix?"
EndDlog
If ok=1 Then
x⟶a
y⟶b
z⟶c
t⟶d
EndIf
EndPrgm
```

Function TF2SS2(sys)

```
TF2SS2(sys)
Func
Local i,t,n,g,a,b,c
expand(sys,s)
jki\poly2cof(getNum(sys),s)⟶n
jki\poly2cof(getDenom(sys),s)⟶g
dim(g)-1⟶t
augment(newList(t+1-dim(n)),n)⟶n
newMat(t,t)⟶a
For i,1,t-1
 1⟶a[i,i+1]
EndFor
For i,1,t
 -g[i+1]⟶a[t,t-i+1]
EndFor
newMat(t,1)⟶b
1⟶b[t,1]
newMat(1,t)⟶c
For i,1,t
 n[i+1]-n[1]*g[i+1]⟶c[1,t-i+1]
EndFor
{string(a),string(b),string(c),string(n[1])}
EndFunc
```

Function ZPKDATA(g)

ZPKDATA(g)

Func

Local d

jki\gettd(g)⟶d

g*e^(d*s)⟶g

{string(jki\roots(jki\poly2cof(getNum(g),s))),string(jki\roots(jki\poly2cof(getDenom(g),s))),jki\dc

gain(g),d}

EndFunc

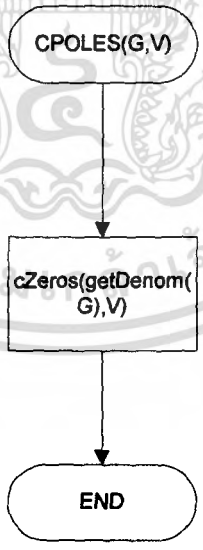
Function CPOLES(g,v)

CPOLES(g,v)

Func

cZeros(getDenom(g),v)

EndFunc



รูปที่ 3.3.2 Flow Chart ของ ฟังก์ชันCPOLES

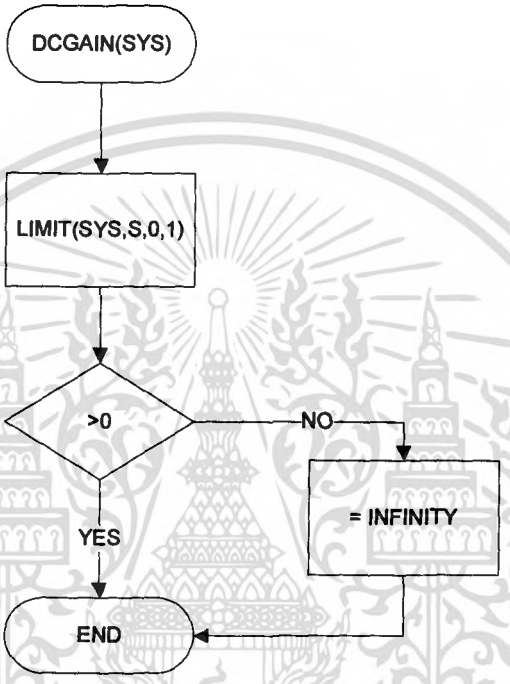
Function DCGAIN(sys)

DCGAIN(sys)

Func

limit(sys,s,0,1)

EndFunc



รูปที่ 3.3.3 Flow Chart ของ ฟังก์ชันDCGAIN

Function PZMAP(g)

PZMAP(g)

Prgm

jki\rlocus(g,{})

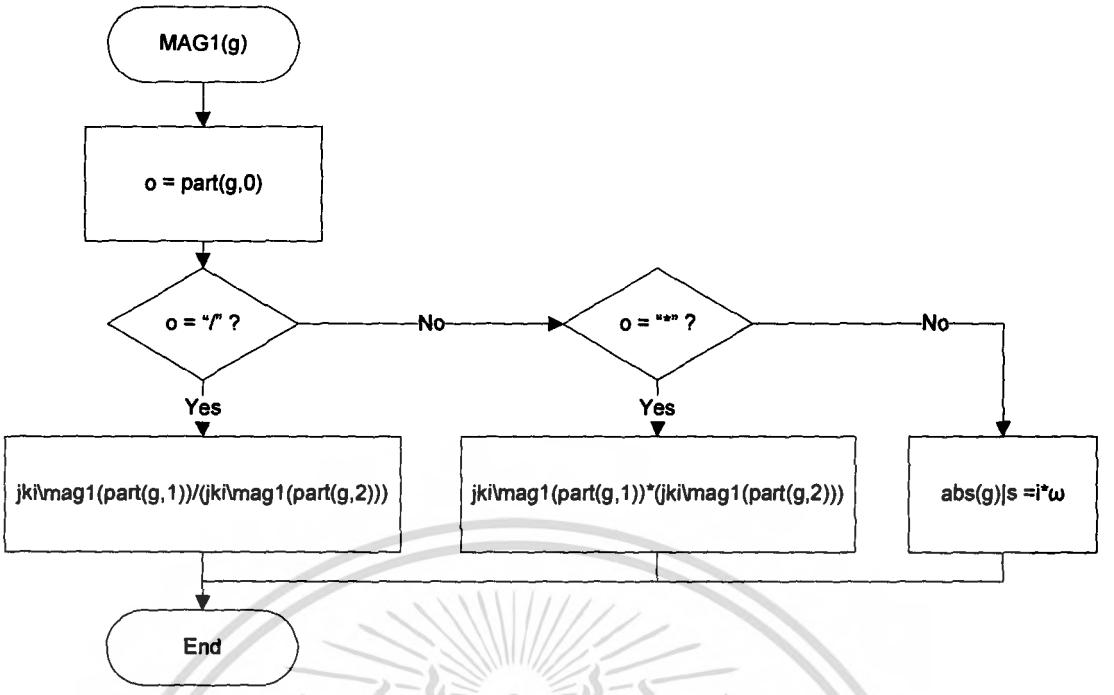
EndPrgm

Function STEP(sys)

```
STEP(sys)
Prgm
factor(sys)*(1/s)⟶sys
jki\InvLap(sys)
EndPrgm
```

Function MAG1(g)

```
MAG1(g)
Func
Local o
part(g,0)⟶o
If o="/" Then
jki\mag1(part(g,1))/(jki\mag1(part(g,2)))
ElseIf o="*" Then
jki\mag1(part(g,1))*jki\mag1(part(g,2))
Else
abs(g)| s=i*ω
EndIf
EndFunc
```



รูปที่ 3.3.4 Flow Chart ของ ฟังก์ชันMAG1

Function MAG(l)

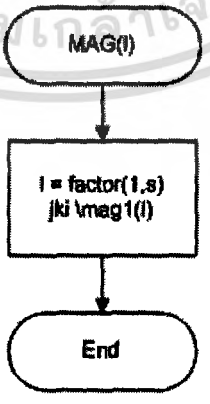
MAG(l)

Func

factor(l,s)⟶l

jki\mag1(l)

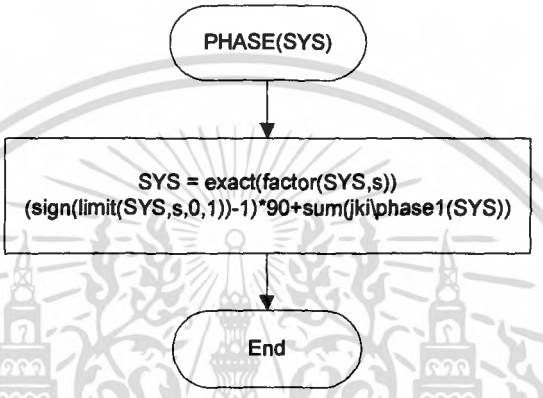
EndFunc



รูปที่ 3.3.5 Flow Chart ของ ฟังก์ชันMAG

Function PHASE(sys)

```
PHASE(sys)
Func
exact(factor(sys,s))⟶sys
(sign(limit(sys,s,0,1))-1)*90+sum(jki\phase1(sys))
EndFunc
```

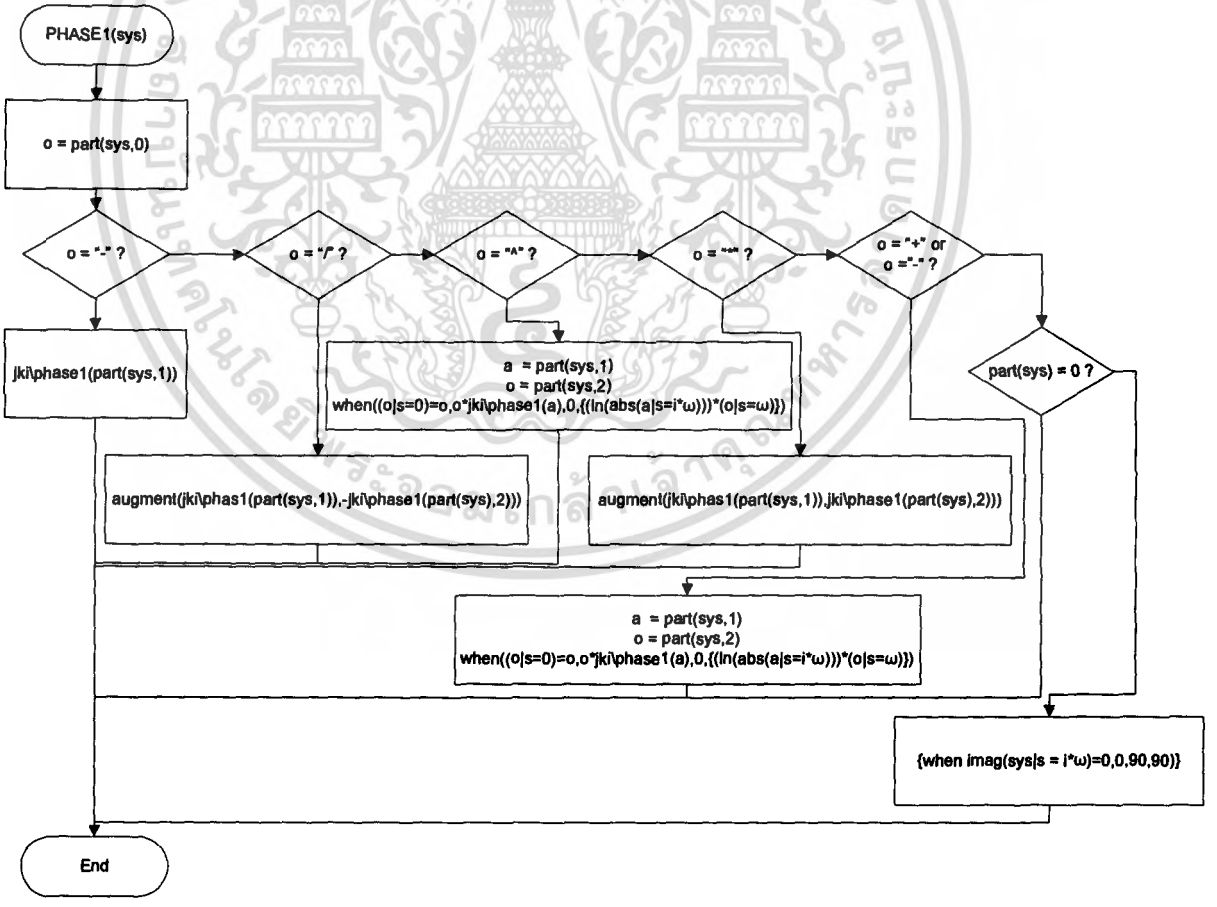


รูปที่ 3.3.6 Flow Chart ของ ฟังก์ชันPHASE

Function PHASE1(sys)

```
PHASE1(sys)
Func
Local a,o
part(sys,0)⟶o
If o="-" Then
 jki\phase1(part(sys,1))
ElseIf o="/" Then
 augment(jki\phase1(part(sys,1)),-jki\phase1(part(sys,2)))
ElseIf o="^" Then
 part(sys,1)⟶a
 part(sys,2)⟶o
 when((o|s=0)=o,o*jki\phase1(a),0,{(ln(abs(a|s=i *ω)))^*(o|s=ω)})
```

```
ElseIf o="*" Then
 augment(jki\phase1(part(sys,1)),jki\phase1(part(sys,2)))
ElseIf o="+" or o="-" Then
 sys|s=i*ω ⟶sys
 real(sys)⟶a
 imag(sys)⟶o
 when(o=0,{(1-sign(a))*90°},0,when(a=0,0,{tan^-1(o/a)},{-tan-1(a/o),90°*sign(o)}|ω>0))
ElseIf
part(sys)=0 Then
 {when(imag(sys|s=i*ω)=0,0,90,90)} °
EndIf
EndFunc
```



รูปที่ 3.3.7 Flow Chart ของ ฟังก์ชันPHASE1

Function POLY2COF(p,v)

```
POLY2COF(p,v)
Func
Local c,n
1⟶n
{p|v=0}⟶c
σ(p,v)⟶p
While when((p|v=∞)0,true,false,true)
augment({(p|v=0)/(n!)},c)⟶c
n+1⟶n
σ(p,v)⟶p
EndWhile
c
EndFunc
```

Function ROOTS(c)

```
ROOTS(c)
Func
Local f,l,z,e,i
dim(c)⟶e
1⟶f
While when(f≤e,c[f]=0,false)
f+1⟶f
EndWhile
If f=e+1:Return {}
e⟶l
While when(l≥1,c[l]=0,false)
l-1⟶l
EndWhile
newList(e-l)⟶z
If f=l:Return z
```

```
list-mat(seq(-c[i]/(c[f]),i,f+1,l))⟶c
l-f-1⟶e
If e>0 Then
 augment(eigVl(augment(c;augment(identity(e),newMat(e,1)))),z)
Else
 augment({c[1,1]},z)
EndIf
EndFunc
```

```
Function ROUTH(poly,s)
ROUTH(poly,s)
Func
Local i,j,px,py,m,t,f
false⟶f
true⟶t
jki\poly2cof(poly,s)⟶poly
dim(poly)⟶px
If px=1
 Return [[poly[1]]]
ceiling(px/2)⟶py
newMat(px,py)⟶m
For i,1,px
 poly[i]⟶m[-mod(i,2)+2,ceiling(i/2)]
EndFor
For i,2,px-1
 1⟶j
 While when(j≤py,when(m[i,j]=0,t,f,f),f)
 j+1⟶j
 EndWhile
 If j>py Then
  For j,1,py
```

```
 (px-i+3-2*j)*m[i-1,j]⟶m[i,j]
 EndFor
ElseIf j≥2 Then
 ε⟶m[i,1]
EndIf
For j,1,py-1
 (m[i-1,j+1]*m[i,1]-m[i-1,1]*m[i,j+1])/(m[i,1])⟶m[i+1,j]
EndFor
EndFor
m
EndFunc
```

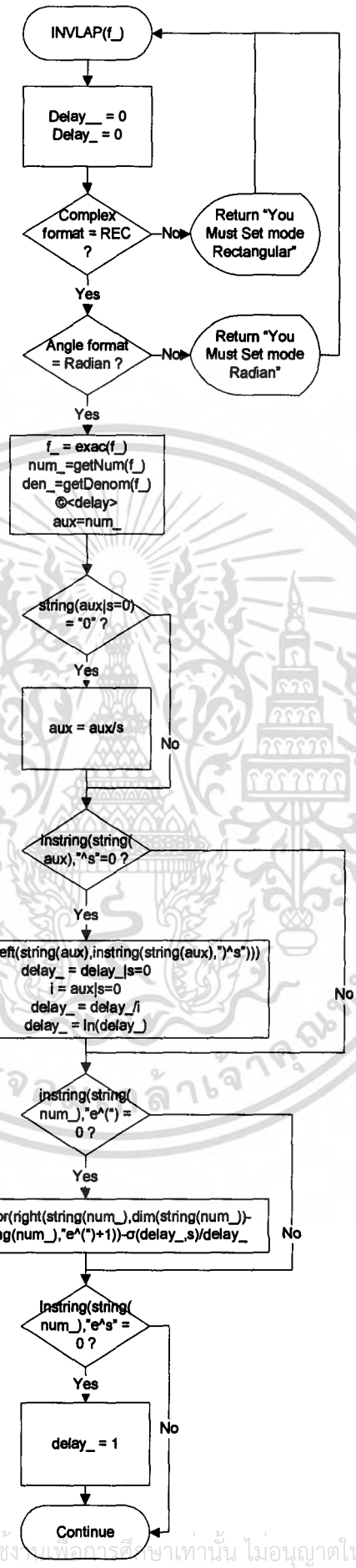
Function INVLAP(f_)

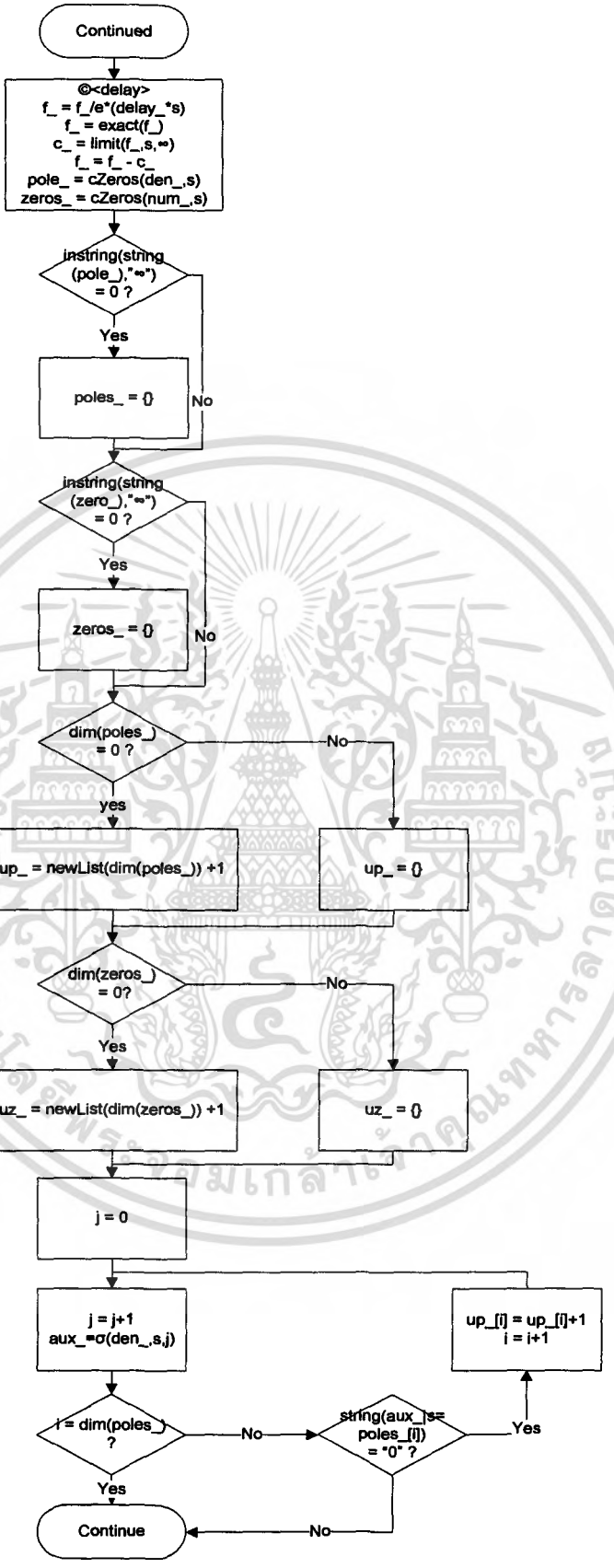
```
INVLAP(f_)
Prgm
Local aux_,poles_,zeros_,cij_,delay__
Local num_,den_,i,j,μp_,μz_,aux,c_
Local delay_
0⟶delay_:0⟶delay__
If getMode("Complex Format")"RECTANGULAR": Return "YOU MUST SET MODE:
Complex Format=RECTANGULAR"
If getMode("Angle")"RADIAN": Return "YOU MUST SET MODE: Angle=RADIAN"
exact(f_)⟶f_
getNum(f_)⟶num_
getDenom(f_)⟶den_
©<delay>
0⟶delay_
num_⟶aux
If string(aux|s=0)="0":aux/s⟶aux
If inString(string(aux),"^s")0 Then
 expr(left(string(aux),inString(string(aux),")^s")))⟶delay_
```

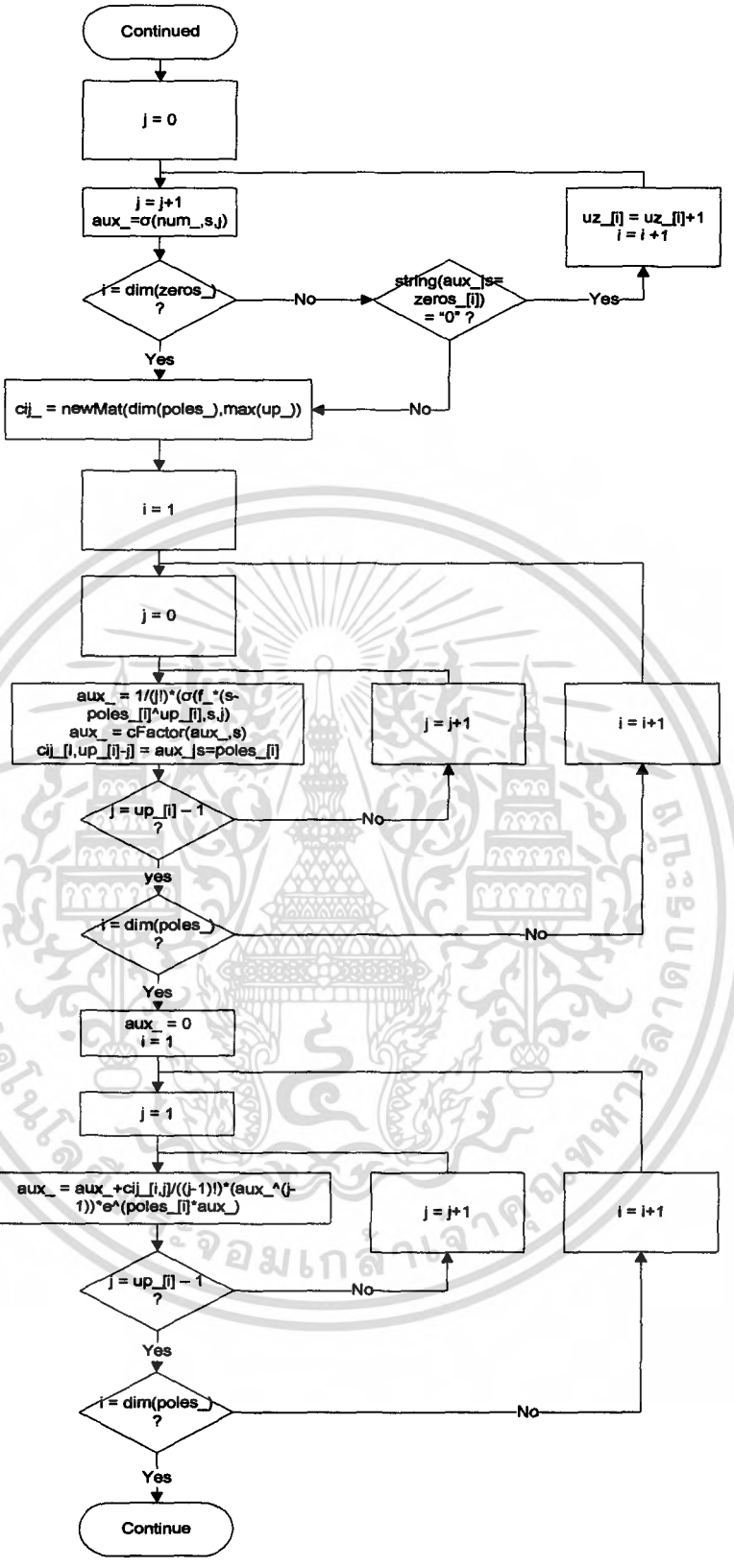
```
 delay_|s=0⟶delay_
 aux|s=0⟶i
 delay_/i⟶delay_
 ln(delay_)⟶delay_
EndIf
If inString(string(num_),"e^(")0 Then
 expr(right(string(num_),dim(string(num_))-inString(string(num_),"e^(")+1))⟶delay_
 σ(delay_,s)/delay_⟶delay_
EndIf
If inString(string(num_),"e^s")0:1⟶delay_
den_⟶aux
If string(aux|s=0)="0":aux/s⟶aux
If inString(string(aux),"^s")0 Then
 expr(left(string(aux),inString(string(aux),")^s")))⟶delay_
 delay_|s=0⟶delay_
 aux|s=0⟶i
 delay_/i⟶delay_
 -ln(delay_)⟶delay_
EndIf
If inString(string(den_),"e^(")0 Then
 expr(right(string(den_),dim(string(den_))-inString(string(den_),"e^(")+1))⟶delay_
 -σ(delay_,s)/delay_⟶delay_
EndIf
If inString(string(den_),"e^s")0:-1⟶delay_
©<\delay>
f_/e^(delay_*s)⟶f_
exact(f_)⟶f_
limit(f_,s,∞)⟶c_
f_-c_⟶f_
cZeros(den_,s)⟶poles_
cZeros(num_,s)⟶zeros_
If inString(string(poles_),"∞")0
```

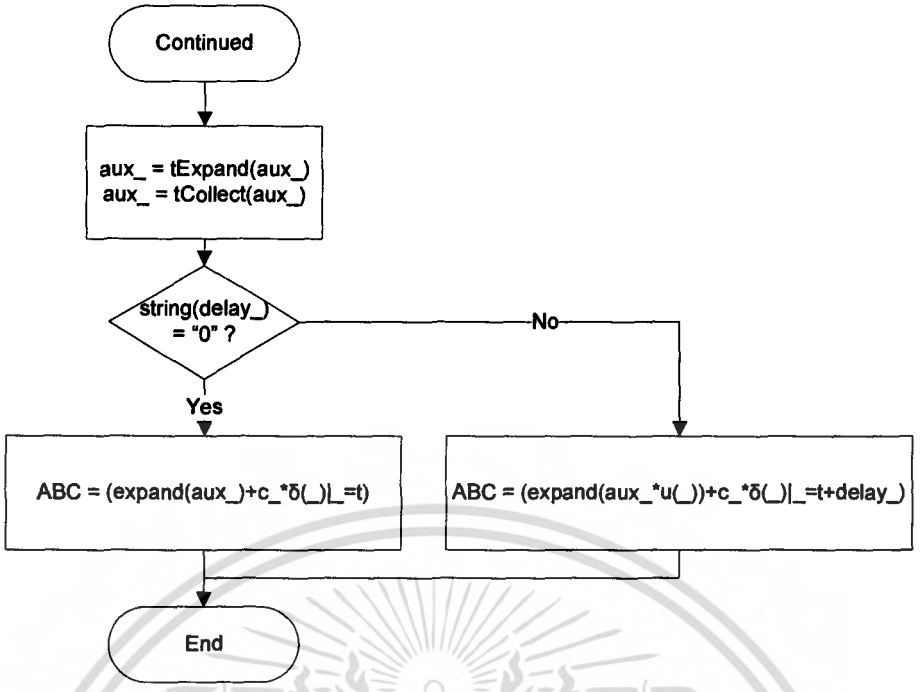
```
  {}⟶poles_
 If inString(string(poles_),"∞")0
  {}⟶zeros_
 If dim(poles_)0 Then
  newList(dim(poles_))+1⟶μp_
 Else
  {}⟶μp_
 EndIf
 If dim(zeros_)0 Then
  newList(dim(zeros_))+1⟶μz_
 Else
  {}⟶μz_
 EndIf
 For i,1,dim(poles_)
  0⟶j
  Loop
   j+1⟶j
   σ (den_,s,j)⟶aux_
   If string(aux_|s=poles_[i])="0" Then
    μp_[i]+1⟶μp_[i]
   Else
    Exit
   EndIf
  EndLoop
EndFor
For i,1,dim(zeros_)
 0⟶j
 Loop
  j+1⟶j
  σ (num_,s,j)⟶aux_
  If string(aux_|s=zeros_[i])="0" Then
   μz_[i]+1⟶μz_[i]
```

```
    Else
     Exit
    EndIf
  EndLoop
 EndFor
 newMat(dim(poles_),max(μp_))⟶cij_
 For i,1,dim(poles_)
  For j,0,μp_[i]-1
   1/(j!)*( σ (f_*(s-poles_[i])^μp_[i],s,j))⟶aux_
   cFactor(aux_,s)⟶aux_
   cFactor(aux_,s)⟶aux_
   aux_|s=poles_[i]⟶cij_[i,μp_[i]-j]
  EndFor
 EndFor
 0⟶aux_
 For i,1,dim(poles_)
  For j,1,μp_[i]
   aux_+cij_[i,j]/((j-1)!)*(_^(j-1))*e^(poles_[i]*_)⟶aux_
  EndFor
 EndFor
 tExpand(aux_)⟶aux_
 tCollect(aux_)⟶aux_
 If string(delay_)="0" Then
  (expand(aux_)+c_*δ(_)|_=t)⟶ABC
 Else
  (expand(aux_*u(_))+c_*δ (_)|_=t+delay_)⟶ABC
 EndIf
 EndPrgm
```









รูปที่ 3.3.8 Flow Chart ของ ฟังก์ชันINVLAP

Function PID(sys)

PID(sys)

Prgm

Local i,j,z

jki\clLoop((k*sys),1)⟶Exp

getDenom(Exp)⟶Exp

jki\routh(Exp,s)⟶Routh

For i,1,rowdim(Routh)

Routh[i,1]⟶a

Try

if solve(a=0,k) = not false then

solve(a=0,k)⟶kcr

endif

else

czeros(a,k)⟶kw

exit

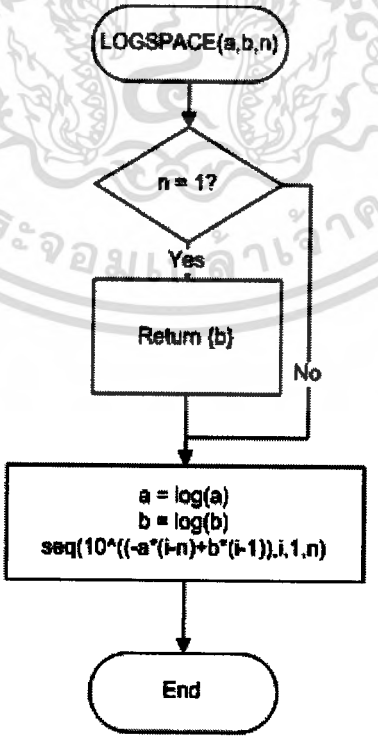
```
  clrerr
 endtry
endfor
if dim(kw)>1 then
for j,1,dim(kw)
 if angle(kw[j])=0 then
  if kw[j]>kcr then
   kw[j]⟶kcr
  endif
 endif
endfor
else
  kw[1]⟶kcr
endif
Exp|k=kcr⟶Exp
Exp|s=i*ω⟶Exp
czeros(Exp,ω)⟶ω1
0⟶ω2
for j,1,dim(ω1)
 if angle(ω1[j])=0 then
  if ω1[j]>ω2 then
   ω1[j]⟶ω2
  endif
 endif
endfor
approx((2*π)/ω2)⟶Pcr
lbl z
0⟶ch
popup {"P-Controller","PI-Controller","PID-Controller","Exit"},ch
If ch=1 then
 Dialog
  Title "P - C O N T R O L L E R"
```

```
 Text  "Kp = " &string(approx(0.5*Kcr))
 Text  "Ti = ∞"
 Text  "Td = " &string(approx(0))
EndDlog
If ok=1 Then
 goto z
else
 DispHome
endif
ElseIf ch=2 then
Dialog
 Title  "P I - C O N T R O L L E R"
 Text  "Kp = " &string(approx(0.45*Kcr))
 Text  "Ti = " &string(approx((1/12)*Pcr))
 Text  "Td = " &string(approx(0))
EndDlog
If ok=1 Then
 goto z
else
 DispHome
endif
ElseIf ch=3 then
Dialog
 Title  "P I D - C O N T R O L L E R"
 Text  "Kp = " &string(approx(0.6*Kcr))
 Text  "Ti = " &string(approx(0.5*Pcr))
 Text  "Td = " &string(approx(0.125*Pcr))
EndDlog
If ok=1 Then
 goto z
else
 DispHome
```

```
 endif
Elseif ch=4 then
 DispHome
endif
EndPrgm
```

Function LOGSPACE(a,b,n)

```
LOGSPACE(a,b,n)
Func
If n=1
 Return {b}
Local i
log(a)⟶a
log(b)⟶b
seq(10^((-a*(i-n)+b*(i-1))/(n-1)),i,1,n)
EndFunc
```



รูปที่ 3.3.9 Flow chart ของฟังก์ชั่น LOGSPACE

```
Function RLOCUS(c,kl)
RLOCUS(c,kl)
Prgm
Local l,m,d,z,p,f,i,o
dim(kl)⟶f
jki\zpkdata(c)⟶d
If d[4]0 Then
 Text "Error: rlocus doesn't support time delay"
 Return
EndIf
StoGDB o
FnOff
setMode({"Exact/Approx","AUTO","Complex Format","RECTANGULAR"})⟶m
setGraph("Axes","On")
expr(d[1])⟶z
expr(d[2])⟶p
p⟶l
jki\poly2cof(getNum(1+k*c),s)⟶c
ClrIO
Disp "Calculating Locus...0%"
For i,1,f
 augment(l,jki\roots(c|k=kl[i]))⟶l
 Output 0,120,string(exact(intDiv(i*100,f)))&"%"
EndFor
augment(l,z)⟶l
Output 0,120,"done!"
ClrDraw
real(l)⟶lr
imag(l)⟶li
NewPlot 1,1,lr,li,4
Try
 ZoomData
```

```
 Else
 If xmax=xmin Then
  xmin-1⟶xmin
  xmax+1⟶xmax
 EndIf
 If ymin=ymax Then
  ymin-1⟶ymin
  ymax+1⟶ymax
 EndIf
 DispG
 ClrErr
EndTry
 For i,1,dim(z)
 PtText "o",real(z[i])-3*Δx,imag(z[i])+4*Δy
EndFor
For i,1,dim(p)
 PtText "x",real(p[i])-3*Δx,imag(p[i])+4*Δy
EndFor
Trace
setMode(m)
PlotsOff
RclGDB o
DelVar lr,li
ClrIO
DispHome
EndPrgm
```

Function ZOOMFIT2()

```
ZOOMFIT2()
Prgm
Try
 ZoomFit
 Else
 If xmax=xmin Then
  xmin-1⟶xmin
  xmax+1⟶xmax
 EndIf
 If ymin=ymax Then
  ymin-1⟶ymin
  ymax+1⟶ymax
 EndIf
 DispG
 ClrErr
EndTry
EndPrgm
```

Function CLLOOP(Ks,Gs)

```
CLLOOP(Ks,Gs)
Func
Ks/(1+(Ks*Gs))
EndFunc
```

****

บทที่ 4
วิธีการใช้งานและตัวอย่างการทำงานของโปรแกรมฟังก์ชันต่างๆ

4.1 วิธีการใช้งานและตัวอย่างการทำงานของโปรแกรมฟังก์ชันต่างๆ

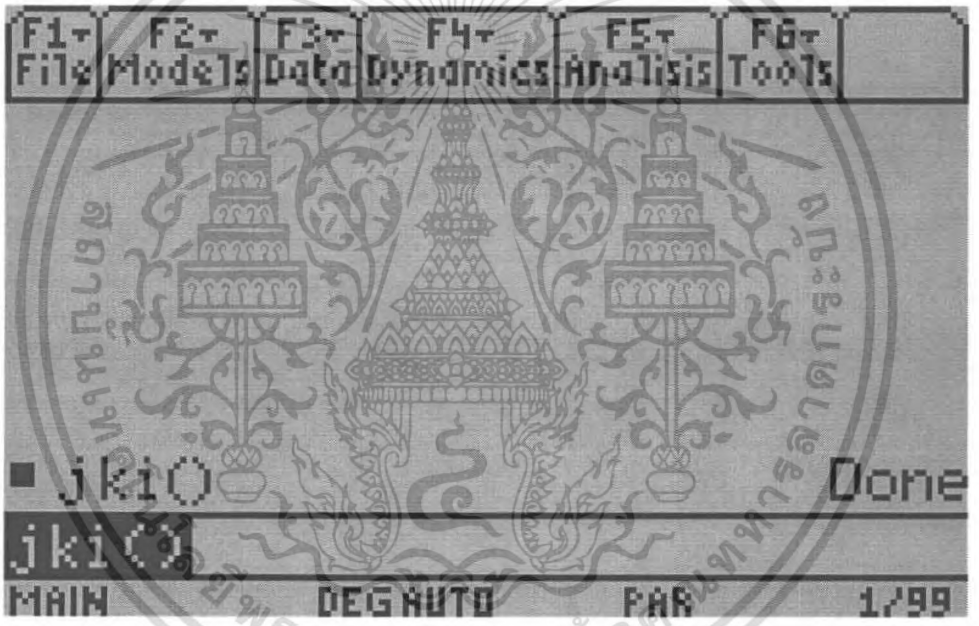
JKI Toolbox เป็นโปรแกรมช่วยในการคำนวณทางคณิตศาสตร์ของระบบควบคุม และการคำนวณทางคณิตศาสตร์สำหรับการปรับแต่งระบบควบคุมแบบพีไอดี (PID) โดยการใช้งานโปรแกรม JKI Toolbox สามารถใช้งานผ่านทางเครื่องคิดเลขรุ่นTI-89โดยการดาวน์โหลดโปรแกรม JKI Toolbox ลงในตัวเครื่องคิดเลขจากเครื่องคอมพิวเตอร์โดยภายในตัวโปรแกรมจะมีฟังก์ชันการใช้งาน ในการคำนวณเฉพาะอย่างทางคณิตศาสตร์ของระบบควบคุม และจะแยกเป็นฟังก์ชันการใช้งานแตกต่างกันออกไป เช่น ฟังก์ชันการคำนวณหาค่าโพล และค่าซีโร่ของระบบควบคุม ฟังก์ชันการแสดงกราฟของตำแหน่งโพล และซีโร่ ของระบบควบคุม ฟังก์ชันการคำนวณหาคำตอบของอินเวอส์ลาปลาส(Inverse Laplace)ฟังก์ชันการหาค่าของ Kp Ti Td และการปรับแต่งค่าของระบบควบคุมแบบพีไอดี (PID) ฟังก์ชันการหาทรานเฟอร์ฟังก์ชัน (Transfer Function) ของระบบ ฟังก์ชันการคำนวณตารางเราธ์-เฮอร์วิธ โดยสามารถเรียกใช้งานฟังก์ชันการคำนวณต่างๆเหล่านี้ผ่านทางฟังก์ชันเมนูของโปรแกรม JKI Toolbox ซึ่งฟังก์ชันต่างๆ ของโปรแกรม JKI Toolbox จะถูกเก็บอยู่ใน Folder JKI และอยู่ในสภาพพร้อมใช้งานโดยโปรแกรม JKI Toolbox มีวิธีใช้ดังนี้

4.1.1 วิธีการใช้งานโปรแกรม JKI Toolbox

JKI( )

เป็นฟังก์ชันที่เรียก เมนู ของ JKI Toolbox ขึ้นมา เพื่อสะดวก ในการใช้ฟังก์ชันต่างๆ โดยไม่ต้องพิมพ์ คำสั่งเรียกฟังก์ชันโดยตรง โดยมีวิธีการเรียกใช้ดังนี้

EX : JKI()



รูปที่ 4.1 ฟังก์ชัน JKI

โดย ที่เมนู สามารถกด Hotkey เพื่อเรียกคำสั่งได้ดังนี้

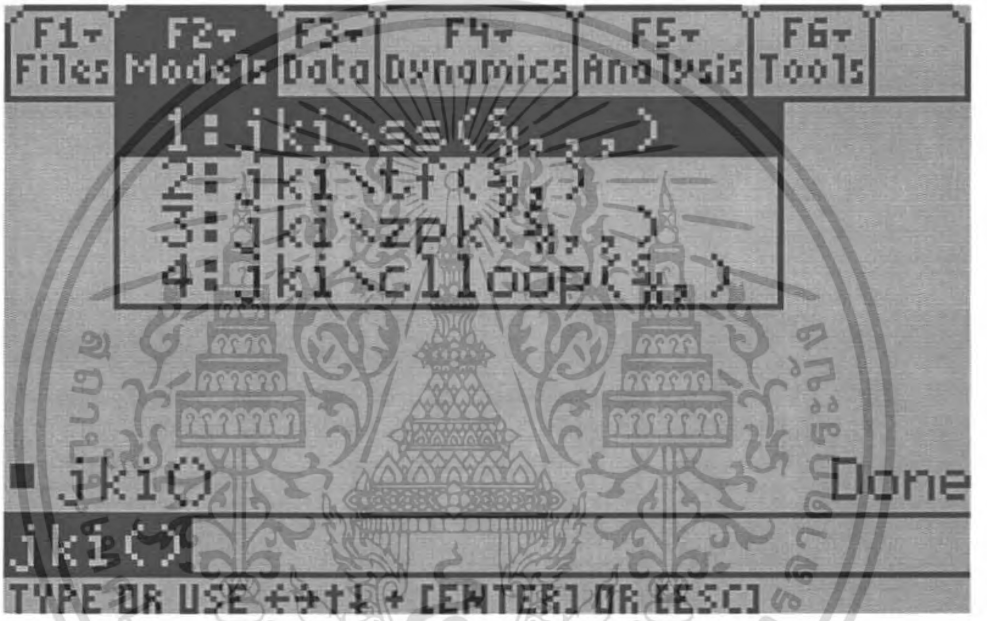
F1-File

- Function ABOUT()
- Function QUIT( )

รูปที่ 4.2 F1 – Menu

F2-Models

- Function SS(A,B,C,D)
- Function TF(Zlist,Plist)
- Function ZPK(Zlist,Plist,K)
- Function CLLOOP(Gs,Hs)



รูปที่ 4.3 F2 - Menu

F3-Data

- Function GETTD(SYS)
- Function TF2SS(SYS)
- Function ZPKDATA(SYS)

รูปที่ 4.4 F3 – Menu

F4-Dynamics

- Function DCGAIN(SYS)
- Function CPOLES(EXPR,VAR)
- Function PZMAP(SYS)

รูปที่ 4.5 F4 - Menu

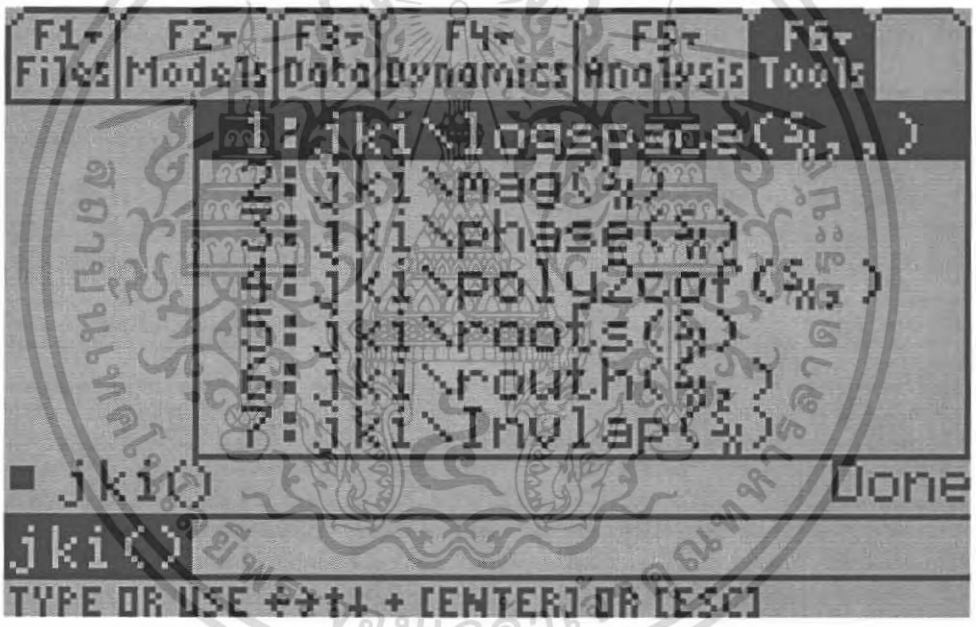
F5-Analysis

- Function STEP(SYS)
- Function RLOCUS(SYS,kList)
- Function PID(SYS)

รูปที่ 4.6 F5 - Menu

F6-Tools

- Function LOGSPACE(F,L,N)
- Function MAG(SYS)
- Function PHASE(SYS)
- Function POLY2COF(POLY)
- Function ROOTS(COEF)
- Function ROUTH (POLY, VAR)
- Function INVLAP(SYS)



รูปที่ 4.7 F6 - Menu

QUIT()

เป็นฟังก์ชันที่ใช้สำหรับ ออกจาก เมนู JKI() ให้เป็น เมนูปกติของ TI-89 โดยมีวิธีการเรียกใช้ดังนี้

EX : QUIT()

รูปที่ 4.8 Function QUIT

SS(A,B,C,D)

เป็นฟังก์ชัน ที่ใช้สำหรับ แปลงค่าจากState Space เมื่อ A,B,C,D คือตัวแปรจากสมการ

$$\dot{x} = Ax + Bu$$

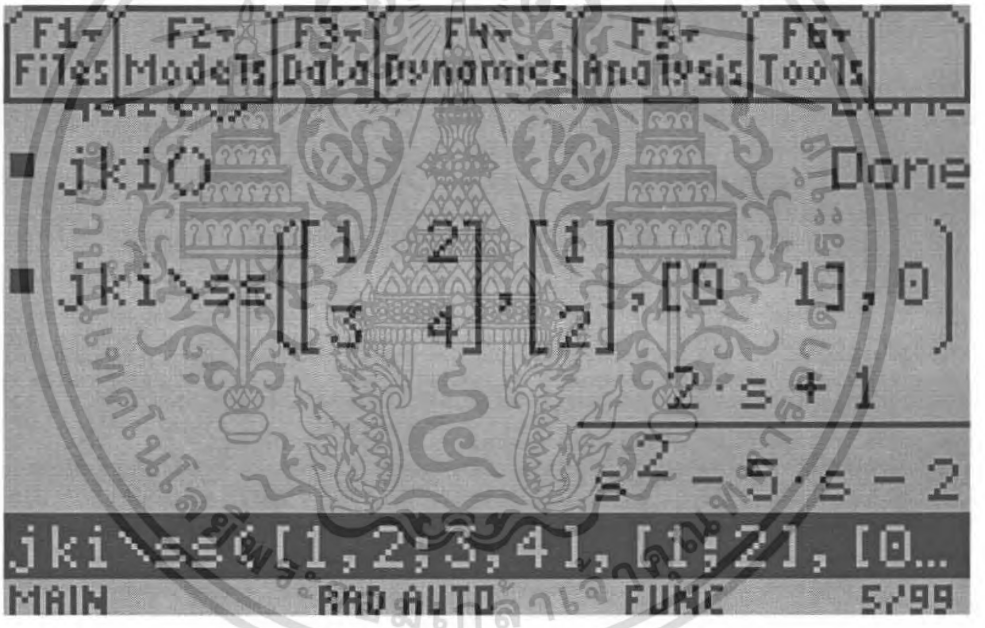
$$y = Cx + Du$$

เป็น Transfer function

โดยมีวิธีการเรียกใช้ดังนี้

EX : jki\SS([1,2;3,4],[1;2],[0,1],0)

ANSWER : (2s+1)/(s^2-5s-2)



รูปที่ 4.9 Function SS

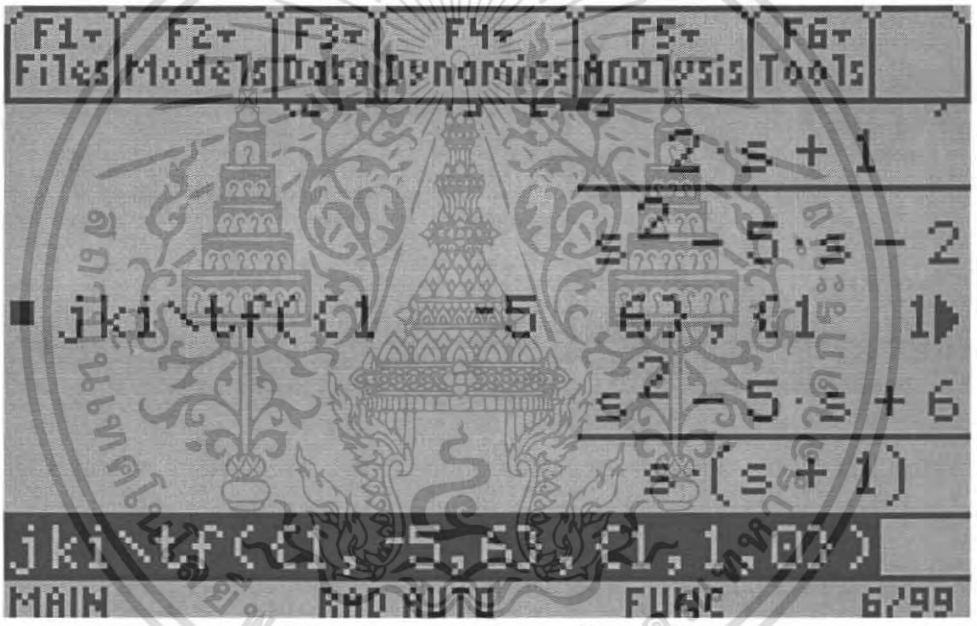
TF(Zlist,Plist)

เป็นฟังก์ชันที่ใช้สำหรับหาค่า Transfer Function จากสัมประสิทธิ์ (Coefficient) โพล และซีโร่ โดย Zlist คือ ซีโร่, Plist คือ โพล

โดยมีวิธีการเรียกใช้ดังนี้

EX : jki\TF({1,-5,6},{1,1,0})

ANSWER : (s^2-5s+6)/s(s+1)



รูปที่ 4.10 Function TF

ZPK(Zlist,Plist,K)

เป็นฟังก์ชันที่ใช้สำหรับนำค่า โพล, ซีโร่, และเกน มาแปลงเป็น Transfer Function โดย Zlist คือค่าของซีโร่, Plist คือค่าของโพล และ K คือค่าของเกน

โดยมีวิธีการเรียกใช้งานดังนี้

EX : jki\ZPK({2,3},{0,-1},5)

ANSWER : (5(s-3)(s-2))/(s(s+1))

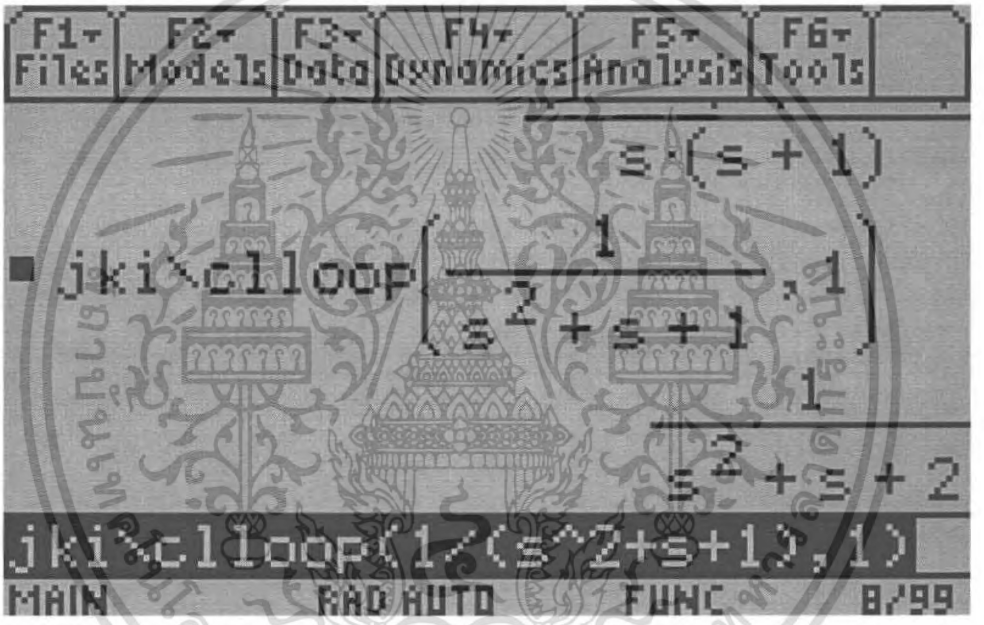
รูปที่ 4.11 Function ZPK

CLLOOP(Gs,Hs)

เป็นฟังก์ชันที่ใช้สำหรับ หา Close Loop ของ Transfer Function ของระบบ Y(S)/U(S) โดยมีวิธีการเรียกใช้ดังนี้

EX : jki\CLLOOP(1/(s^2+s+1),1)

ANSWER : 1/(s^2+s+2)



รูปที่ 4.12 Function CLLOOP

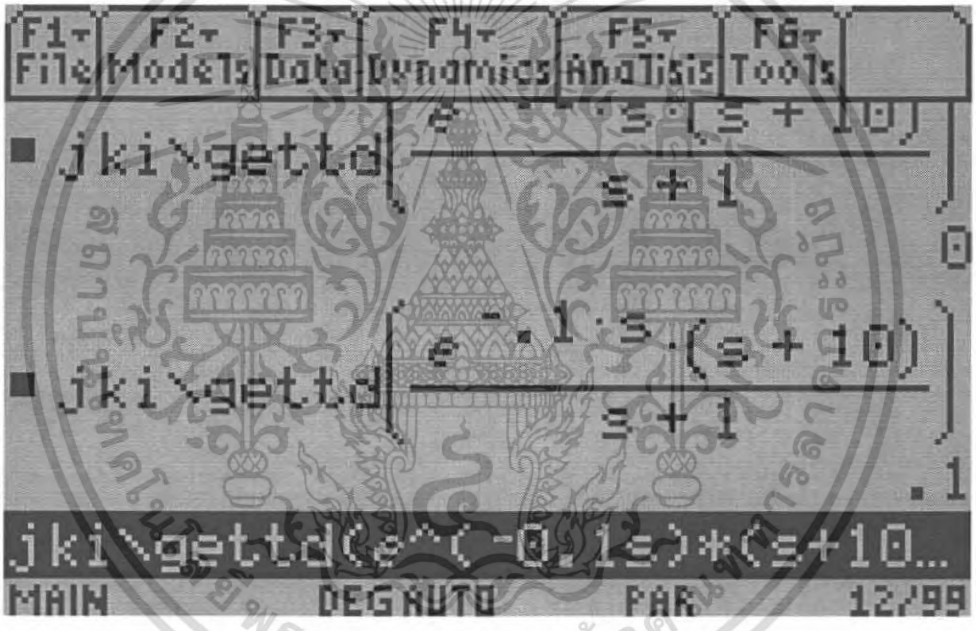
GETTD(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ หาค่า Time Delay ของระบบ โดย SYS คือ Transfer Function ของระบบ

โดยมีวิธีการเรียกใช้ดังนี้

EX: jki\GETTD(e^(-0.1s)*(s+10)/(s+1))

ANSWER : 0.1



รูปที่ 4.13 Function GETTD

TF2SS(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ แปลง Transfer Function เป็น State Space โดย SYS คือ Transfer Function ของระบบ

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

ซึ่งค่าที่ได้จะเป็น เมตริกซ์ A,B,C,D ใน Controllable Canonical Form

โดยมีวิธีการเรียกใช้ดังนี้

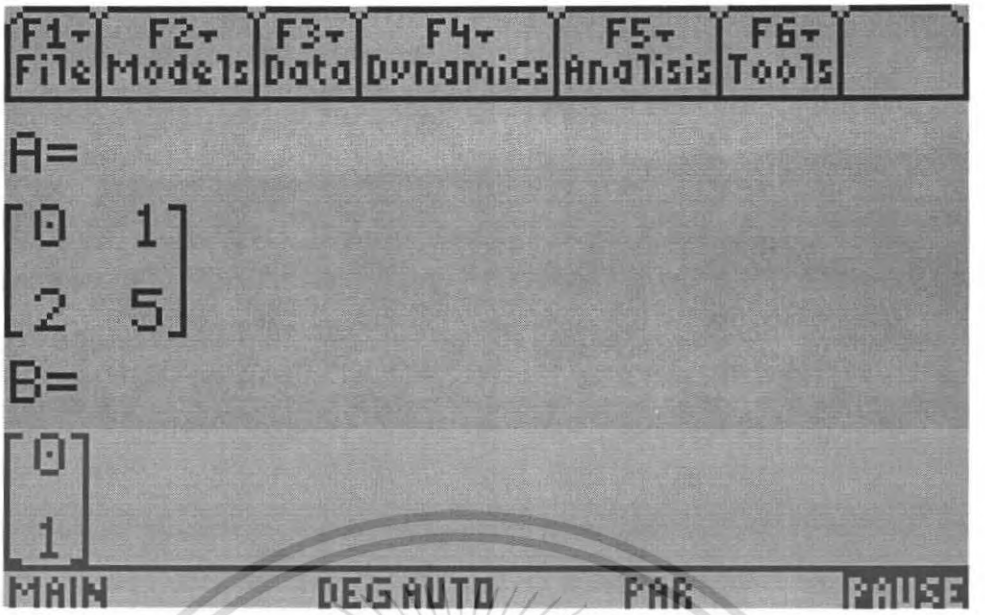
EX : jki\TF2SS((2*s+1)/(s^2-5*s-2))

ANSWER : A = [[0,1][2,5]]

B = [[0][1]]

C = [[1,2]]

D = 0



รูปที่ 4.14 Function TF2SS

ZPKDATA(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ หาค่าโพล, ซีโร่, เกน และ Time Delay โดย SYS คือ Transfer Functionของระบบ

โดยมีวิธีการเรียกใช้ดังนี้

EX : jki\ZPKDATA(e^(-0.2s)*(2*s+1)/(s^2+6*s+10))

ANSWER : {"{-1/2}","{-3-i,-3+i}",1/10,1/5}

รูปที่ 4.15 Function ZPKDATA

DCGAIN(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ หาค่า DC Gain ของระบบ หรือ ค่าสุดท้ายที่ระบบจะเข้าสู่สภาวะคงตัว เมื่อ เวลาเข้าสู่ Infinity โดย SYS คือ Transfer Function ของระบบโดยมีวิธีการเรียกใช้ดังนี้

EX : jki\DCGAIN((s+10)/(s+1))

ANSWER : 10

รูปที่ 4.16 Function DCGAIN

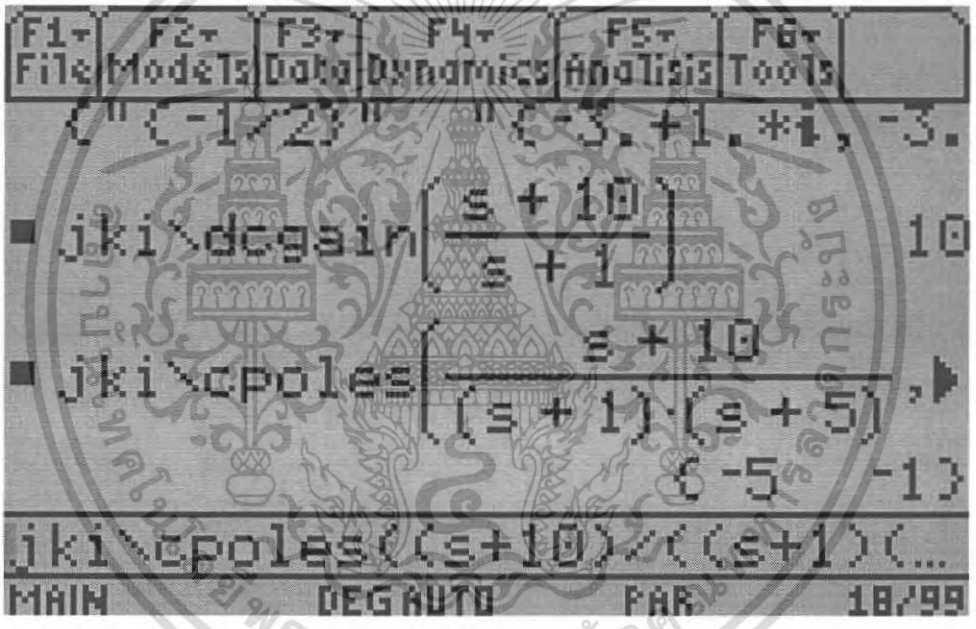
CPOLES(EXPR,VAR)

เป็นฟังก์ชันที่ใช้สำหรับ หาค่า โพล ของระบบ โดย EXPR คือ Transfer Function ของ ระบบ และ VAR คือ ตัวแปรของ Transfer Function

โดยมีวิธีการเรียกใช้ดังนี้

EX : jki\CPOLES((s+10)/((s+1)(s+5)),s)

ANSWER : {-1,-5}

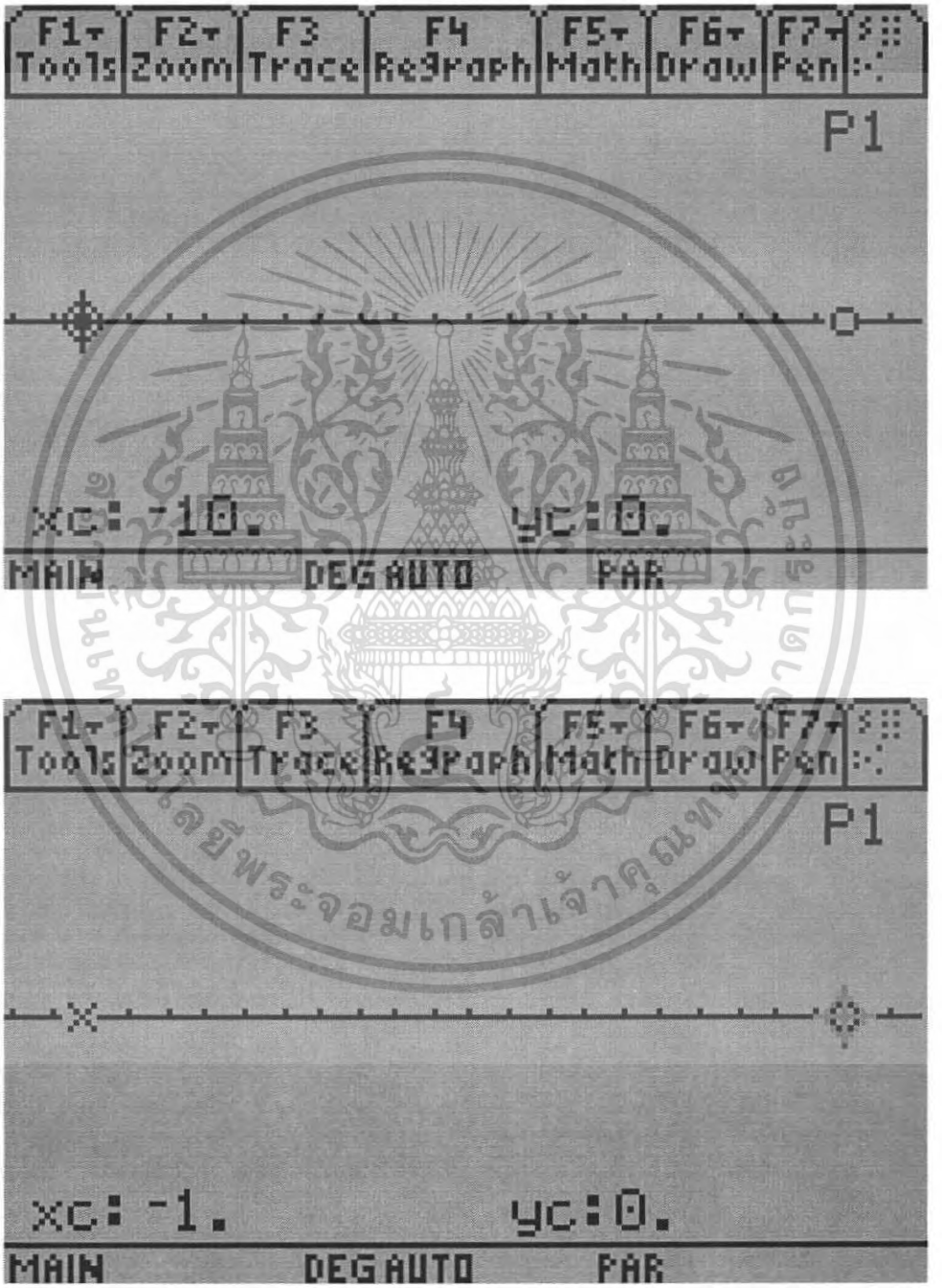


รูปที่ 4.17 Function CPOLES

PZMAP(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ วาดกราฟของ ตำแหน่ง ของ โพล และ ซีโร่ ทั้งหมด บนแกนจริง กับแกนจินตภาพ โดย SYS คือ Transfer Function ของระบบโดยมีวิธีการเรียกใช้ดังนี้

EX : jki\PZMAP((s+1)/(s+10)) (โพลอยู่ที่ 10-และ ซีโร่อยู่ที่ (1-



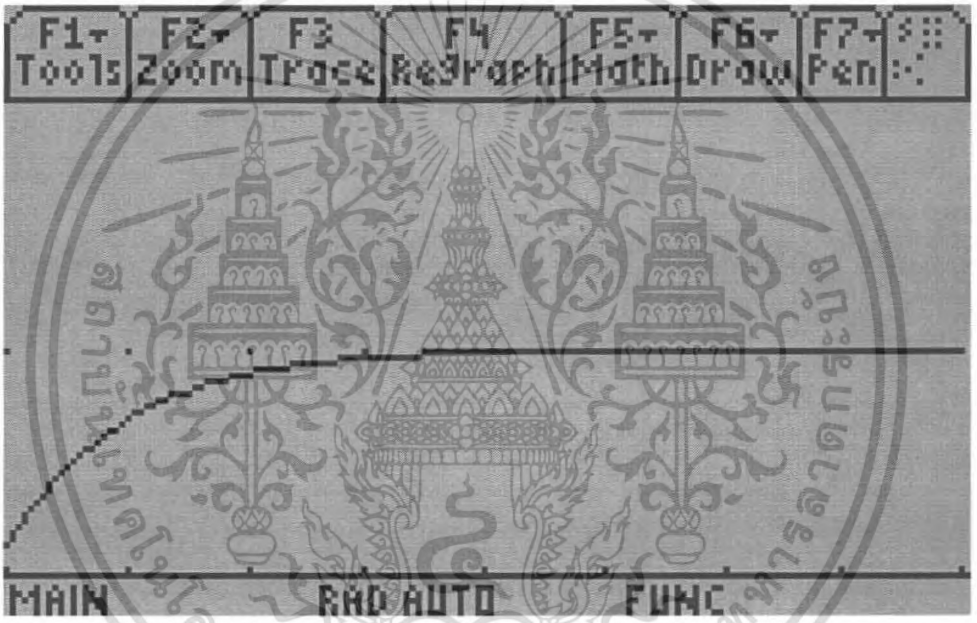
รูปที่ 4.18 Function PZMAP

STEP(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ หาค่า Step Response ที่มาจาก การป้อนสัญญาน Step ซึ่งอยู่ในรูป Time Domain โดย SYS คือ Transfer Function ของระบบโดยมีวิธีการเรียกใช้ดังนี้

EX : jki\STEP(1/(S+1))

ANSWER : 1-e^(-t)

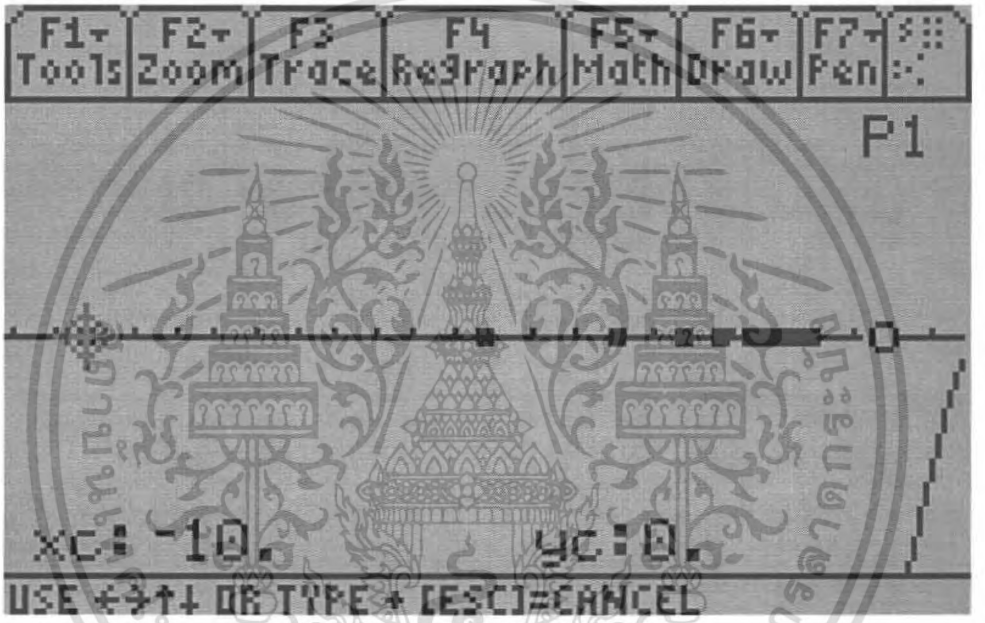


รูปที่ 4.19 Function STEP

RLOCUS(SYS,kList)

เป็นฟังก์ชันที่ใช้สำหรับ วาดกราฟ Root Locus ของระบบ โดย SYS คือ Transfer Function ของระบบ และ kList คือ ค่าของ K ที่ต้องการ plot โดยมีวิธีการเรียกใช้ดังนี้

EX : jki\RLOCUS((s+1)/(s+10))



รูปที่ 4.20 Function RLOCUS

PID(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ การหาค่า Kp, Ti, Td ในการปรับค่า Controller ซึ่งแบ่งเป็นสามแบบ คือ

1. P-controller
2. PI-controller
3. PID-controller

โดย SYS คือ Transfer Function ของระบบ

โดยมีวิธีการเรียกใช้ดังนี้

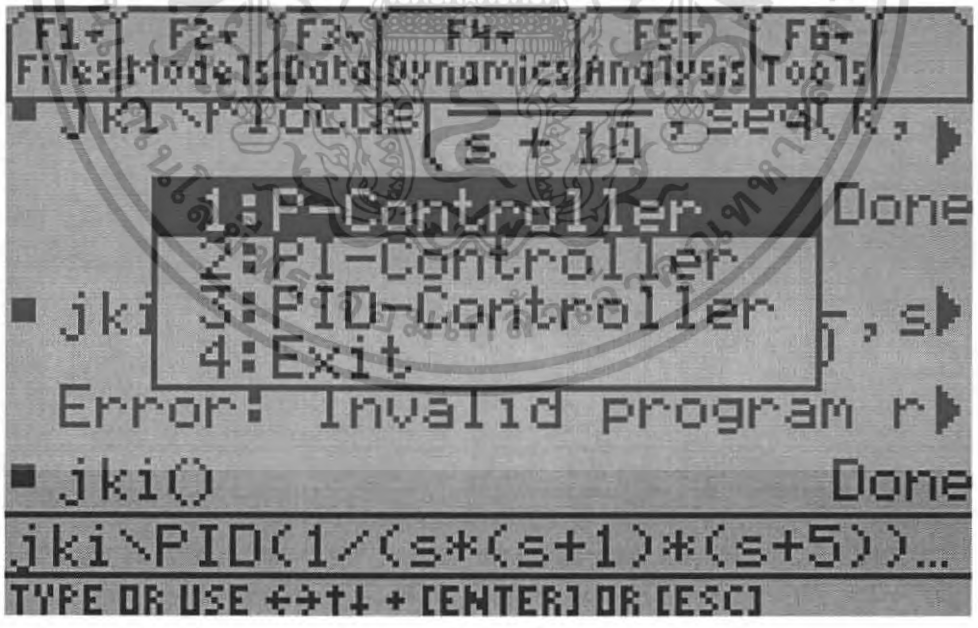
EX : jki\PID(1/(s(s+1)/(s+5)))

ANSWER :

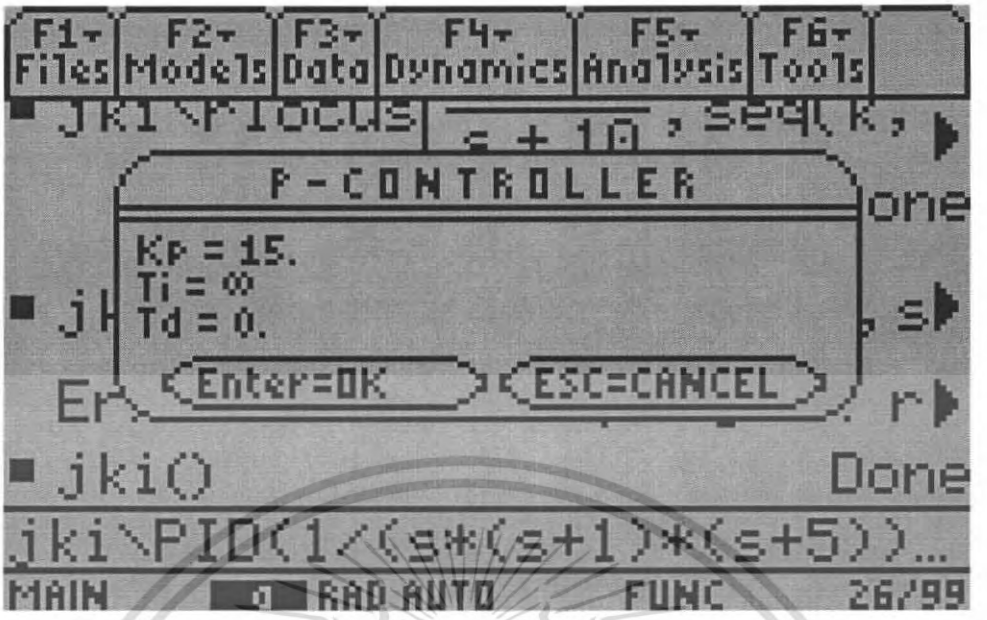
P-Controller : Kp = 15 Ti = ∞ Td = 0

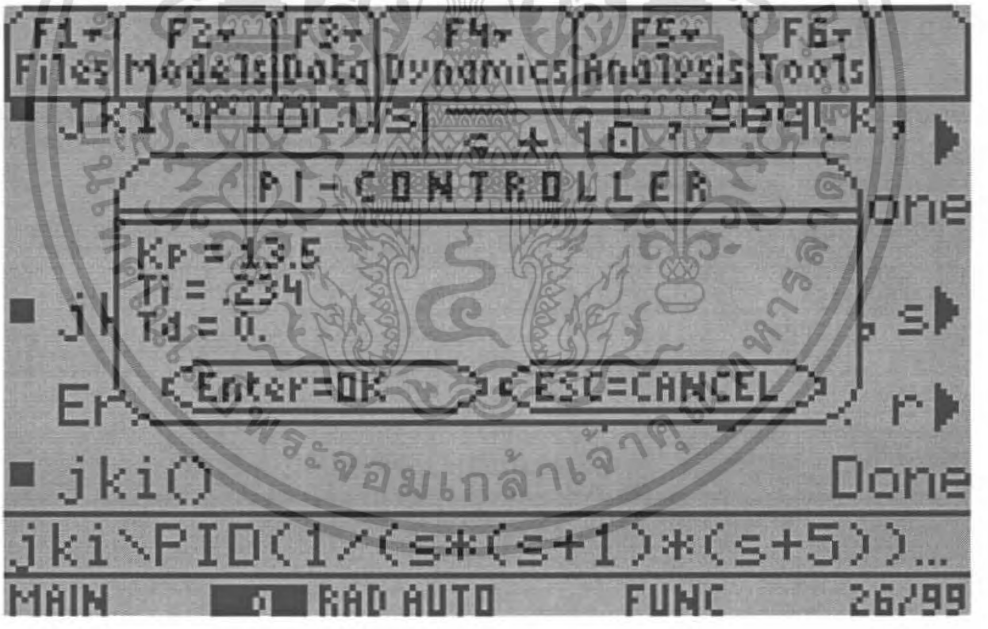
PI-Controller : Kp = 13.5 Ti = 0.234 Td = 0

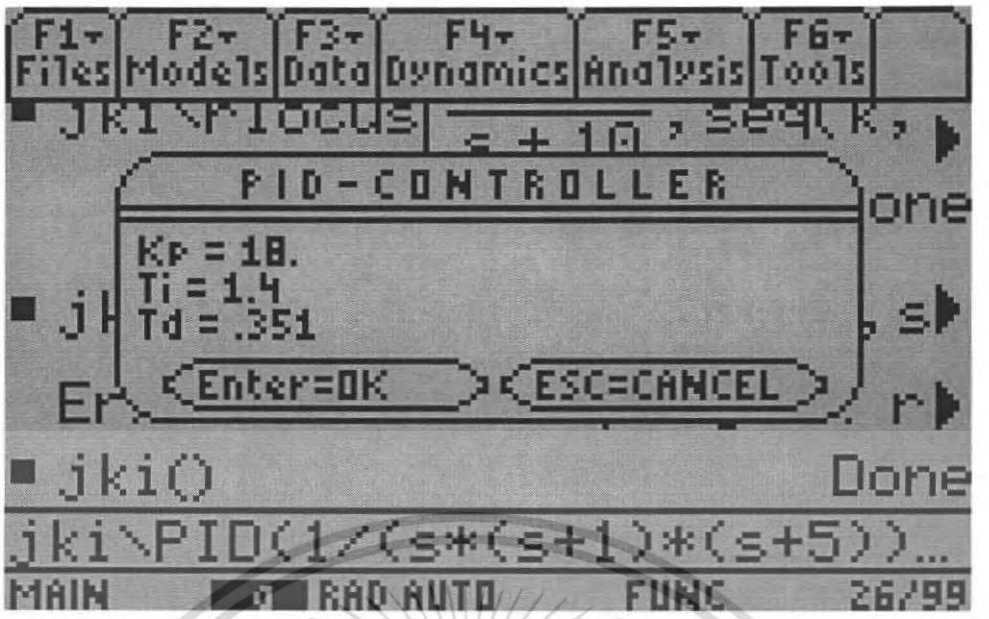
PID-Controller : Kp = 18 Ti = 1.4 Td = 0.351



\







รูปที่ 4.21 Function PID

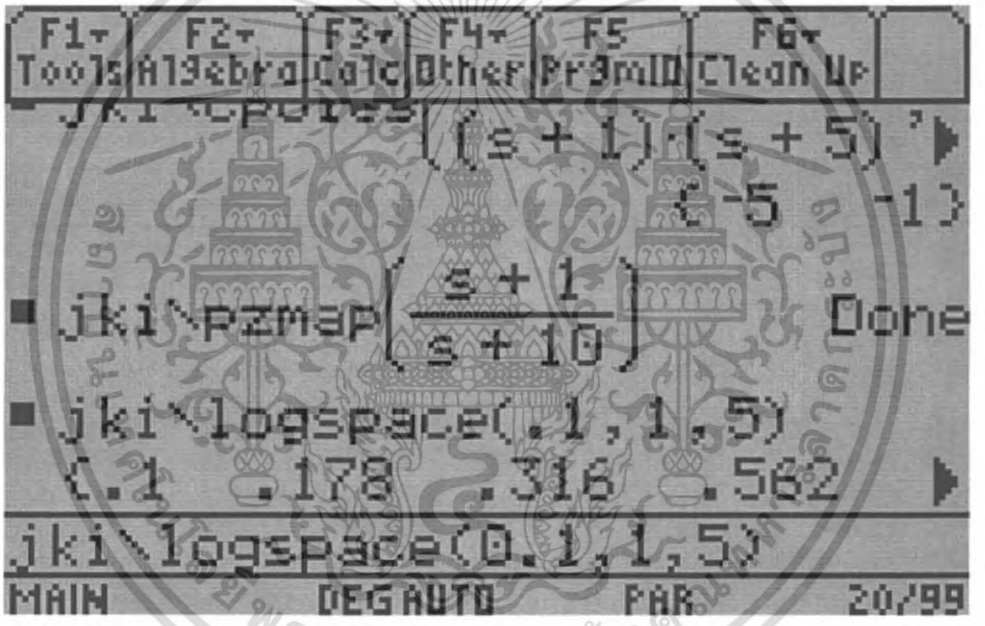
LOGSPACE(F,L,N)

เป็นฟังก์ชันที่ใช้สำหรับ กำหนดค่าของ กราฟ LOG โดย F คือ ค่าความถี่เริ่มต้น L คือ ค่าความถี่สุดท้าย และ N คือ จำนวนช่วงของกราฟ

โดยมีวิธีการเรียกใช้ดังนี้

EX : jki\LOGSPACE(0.1,1,5)

ANSWER : {0.1, 0.177828, 0.316228, 0.562341, 1}



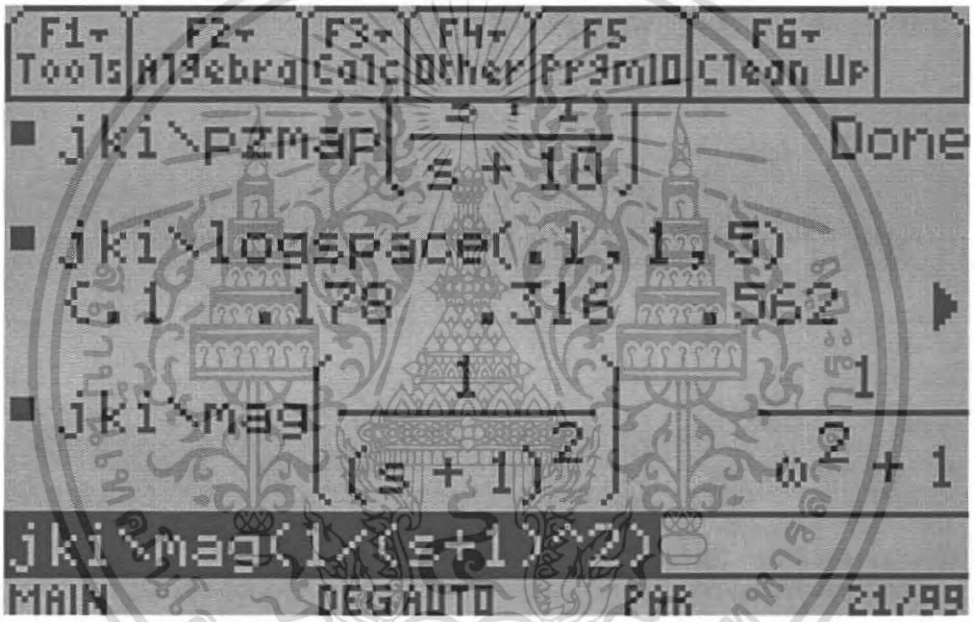
รูปที่ 4.22 Function LOGSPACE

MAG(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ หาค่า Magnitude ของ Transfer Function ของระบบ ซึ่งนำเสนอเป็นค่า omega โดย SYS คือ Transfer Function ของระบบโดยมีวิธีการเรียกใช้ดังนี้

EX : jki\MAG(1/(s+1)^2)

ANSWER : 1/(omega^2+1)



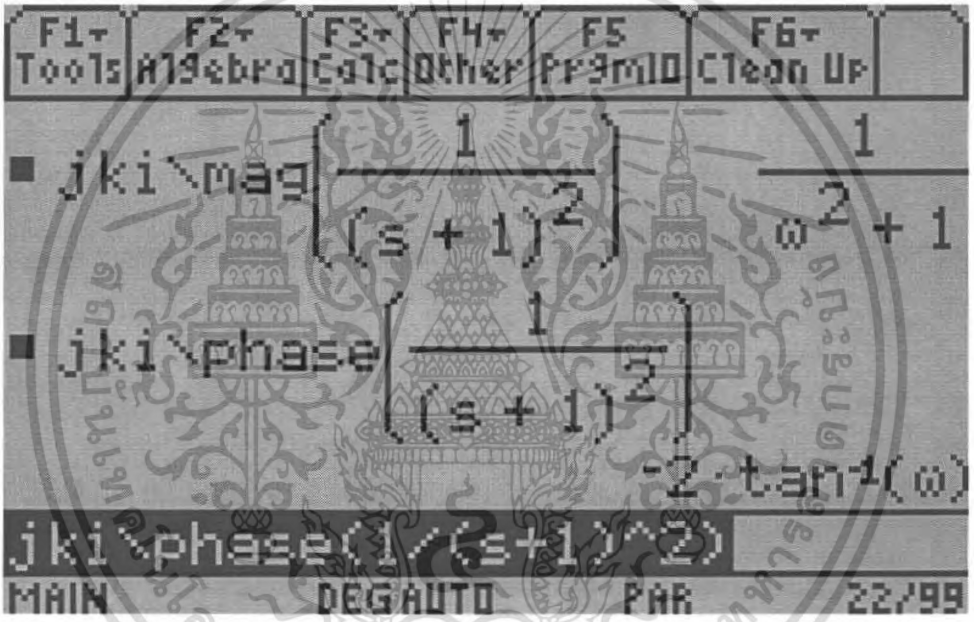
รูปที่ 4.23 Function MAG

PHASE(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ หาค่า Phase ของ Transfer Function ของระบบ ซึ่งนำเสนอเป็นค่า omega โดย SYS คือ Transfer Function ของระบบโดยมีวิธีการเรียกใช้ดังนี้

EX : jki\PHASE(1/(s+1)^2)

ANSWER : $-2*\tan^{-1}(\text{omega})$



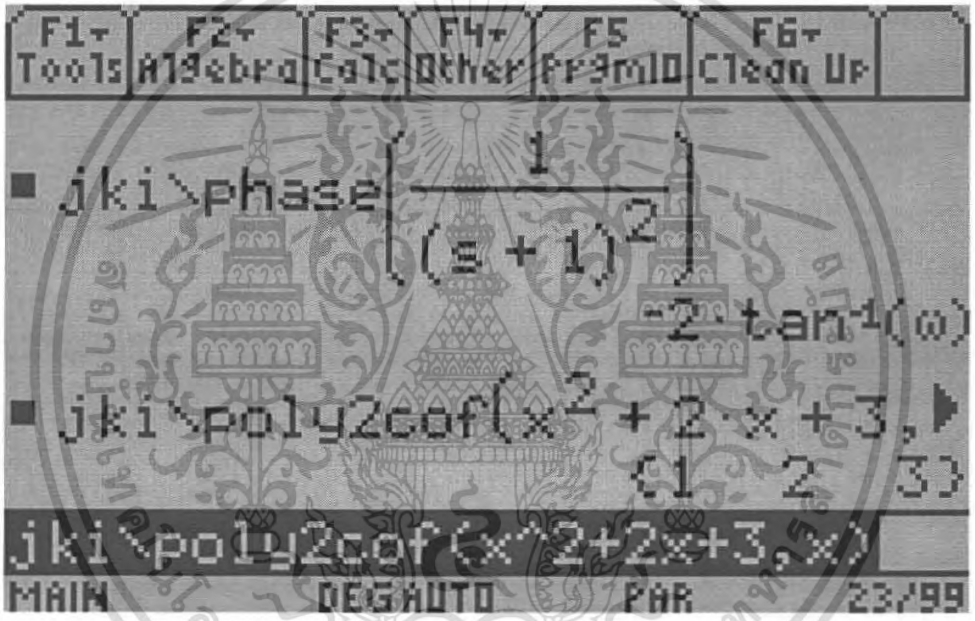
รูปที่ 4.24 Function PHASE

POLY2COF(POLY,VAR)

เป็นฟังก์ชันที่ใช้สำหรับหาค่าสัมประสิทธิ์จากสมการพหุนามโดย POLY คือ สมการพหุนาม , VAR คือ ตัวแปรของสมการพหุนามโดยมีวิธีการเรียกใช้ดังนี้

EX : jki\POLY2COF(x^2+2x+3,x)

ANSWER : {1,2,3}



รูปที่ 4.25 Function POLY2COF

ROOTS(COEF)

เป็นฟังก์ชันที่ใช้สำหรับ คำตอบของสมการพหุนาม จากสัมประสิทธิ์ของสมการพหุนาม โดย COEF คือ สัมประสิทธิ์ของสมการพหุนามโดยมีวิธีการเรียกใช้ดังนี้

EX : jki\ROOTS({1,2,1})

ANSWER : {-1,-1}

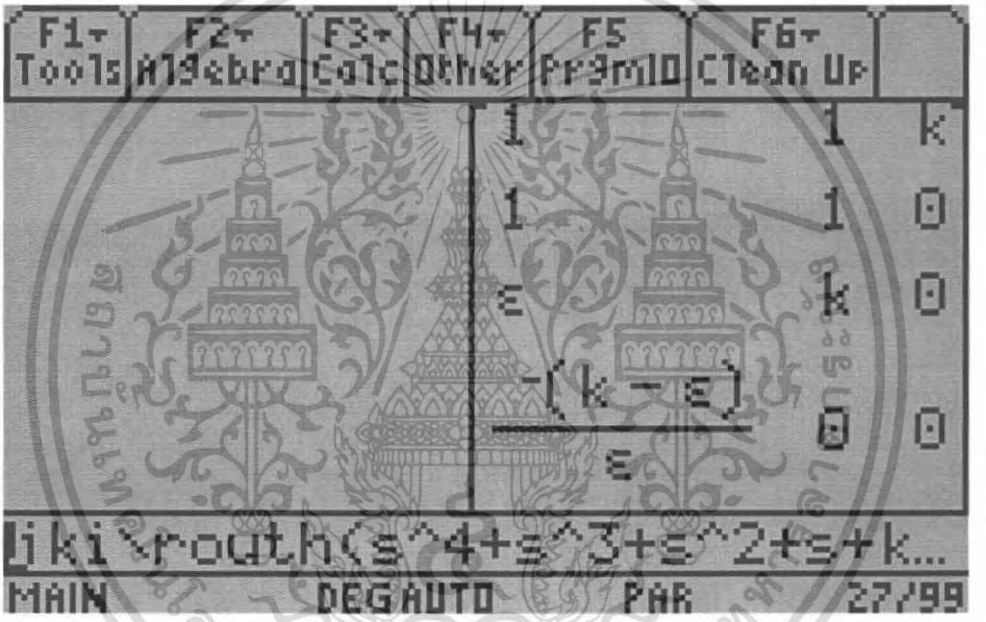
รูปที่ 4.26 Function ROOTS

ROUTH(POLY,VAR)

เป็นฟังก์ชันที่ใช้สำหรับ สร้างตาราง Routh-Hurwitz เป็นเมตริกซ์ โดย POLY คือ สมการพหุนาม ,VAR คือ ตัวแปรของสมการพหุนามโดยมีวิธีการเรียกใช้ดังนี้

EX : jki\ROUTH (s^4+s^3+s^2+s+k,s)

ANSWER : [[1,1,k][1,1,0][eps,k,0][(eps-k)/eps,0,0][k,0,0]]

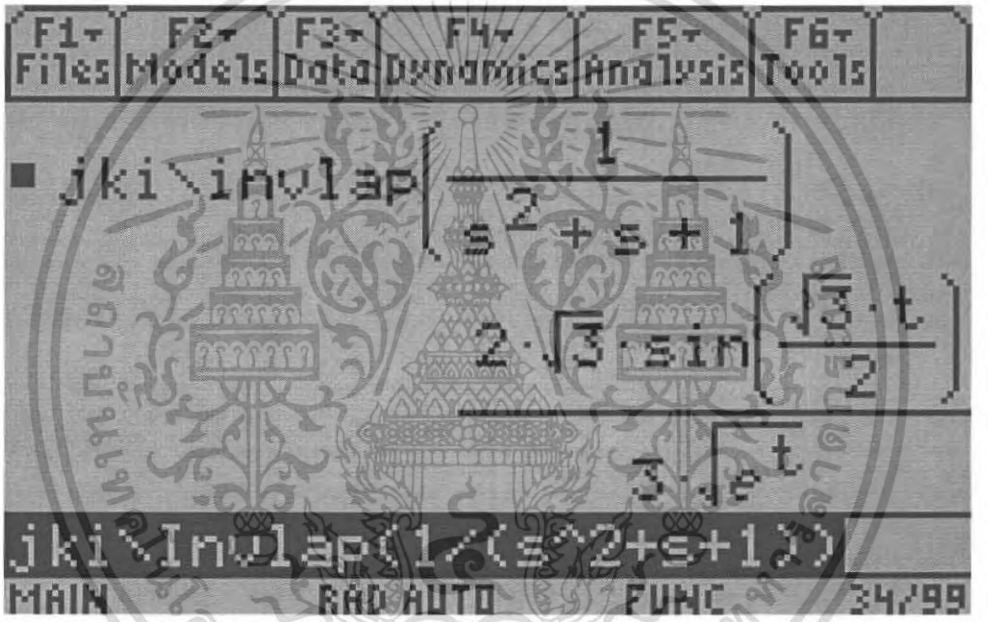


รูปที่ 4.27 Function ROUTH

INVLAP(SYS)

เป็นฟังก์ชันที่ใช้สำหรับ การ Inverse Laplace จาก S-Domain เป็น Time-domain ในตัวแปร t โดยมีวิธีการเรียกใช้ดังนี้

EX : jki\INVLAP(1/(s^2+s+1))



รูปที่ 4.28 Function INVLAP

บทที่ 5

การทดลองและผลการทดลอง

5.1 ผลจากการทดลองการทำงานของโปรแกรม

จากการทดลองการใช้งานโปรแกรมJKI Toolbox โดยการดาวน์โหลดโปรแกรมลงในเครื่องคิดเลขรุ่น Ti89 ผ่านทางเครื่องคอมพิวเตอร์และนำแต่ละฟังก์ชั่นของโปรแกรม JKI Toolbox ไปทดสอบการใช้งานโดยการคำนวนหาคำตอบและผลตอบสนองของสมการในเชิงระบบควบคุมเพื่อหาแต่ละค่าของผลตอบสนองที่ต้องการจากการทำงานของฟังก์ชั่นที่ถูกออกแบบและนำไปทดสอบการคำนวณทางคณิตศาสตร์ในการปรับแต่งระบบควบคุมแบบพีไอดี (PID)จากการทดสอบการทำงานของโปรแกรม JKI Toolbox โดยนำโปรแกรม JKI Toolbox ไปใช้งานในการคำนวณหาผลลัพธ์ในแต่ละฟังก์ชันการใช้งาน และนำโปรแกรม JKI Toolbox ไปทดสอบการใช้งานจริงในการปรับแต่งระบบควบคุมแบบพีไอดี(PID) ของระบบควบคุมจริงจะได้ผลการทดลองดังนี้

ผลจากการทดลองการทำงานของโปรแกรม JKI Toolbox ในการคำนวณหาผลลัพธ์ทางวิศวกรรมระบบควบคุม

Function JKI( )

ผลการทำงานของฟังก์ชันจะเรียกเมนูของ JKI Toolbox ขึ้นมาเพื่อสะดวก ในการใช้ฟังก์ชันต่างๆ โดยไม่ต้องพิมพ์คำสั่งเรียกฟังก์ชันโดยตรง

Function QUIT()

ผลการทำงานของฟังก์ชันจะออกจาก เมนู JKI() ให้เป็น เมนูปกติของ TI-89

Function TF(Zlist,Plist)

ผลการทำงานของฟังก์ชันจะแปลงสัมประสิทธิ์ (Coefficient) ของซีโร่ และ โพล เป็น Transfer Function โดย Zlist คือ สัมประสิทธิ์ ของสมการซีโร่, Plist คือ สัมประสิทธิ์ ของสมการโพล

Function SS(A,B,C,D)

ผลการทำงานของฟังก์ชันจะแปลงค่าจาก State Space เมื่อ A,B,C,D คือตัวแปรจากสมการ

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

เป็น Transfer function

Function ZPK(Zlist,Plist,K)

ผลการทำงานของฟังก์ชันจะนำค่า โพล, ซีโร่, และ เกน มาแปลงเป็น Transfer Function โดย Zlist คือ ค่าของ ซีโร่ ทั้งหมด, Plist คือ ค่าของโพลทั้งหมด และ K คือค่าของเกน

Function GETTD(SYS)

ผลการทำงานของฟังก์ชันจะหาค่า Time Delay ของ ระบบ โดย SYS คือ Transfer Function ของ ระบบ

Function TF2SS(SYS)

ผลการทำงานของฟังก์ชันจะแปลง Transfer Function เป็น State Space โดย SYS คือ Transfer Function ของ ระบบ

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

ซึ่งค่าที่ได้จะเป็น เมตริกซ์ A,B,C,D ใน Controllable Canonical Form

Function ZPKDATA(SYS)

ผลการทำงานของฟังก์ชันจะหาค่า โพล, ซีโร่, เกน และ Time Delay โดย SYS คือ Transfer Function ของ ระบบ

Function DCGAIN(SYS)

ผลการทำงานของฟังก์ชันจะหาค่า DC Gain ของระบบ หรือ ค่าสุดท้ายที่ระบบจะเข้าสู่สภาวะคงตัว เมื่อ เวลาเข้าสู่ Infinity โดย SYS คือ Transfer Function ของ ระบบ

Function CPOLES(EXPR,VAR)

ผลการทำงานของฟังก์ชันจะคำนวนหาค่า โพล ของระบบ โดย EXPR คือ Transfer Function ของระบบ และ VAR คือ ตัวแปรของ Transfer Function

Function PZMAP(SYS)

ผลการทำงานของฟังก์ชันจะวาดกราฟของ ตำแหน่งโพล และ ซีโร่ ทั้งหมด บนแกนจริง กับ แกนจินตภาพ โดย SYS คือ Transfer Function ของ ระบบ

Function STEP(SYS)

ผลการทำงานของฟังก์ชันจะคำนวนหาค่า Step Response ที่มาจากการป้อนสัญญาน Step ซึ่งอยู่ในรูป Time Domain โดย SYS คือ Transfer Function ของระบบ

Function LOGSPACE(F,L,N)

ผลการทำงานของฟังก์ชันจะกำหนดค่าของกราฟ LOG โดย F คือ ค่าความถี่เริ่มต้น L คือ ค่าความถี่สุดท้าย และ N คือ จำนวนช่วงของกราฟ

Function MAG(SYS)

ผลการทำงานของฟังก์ชันจะคำนวนหาค่า Magnitude ของ Transfer Function ของระบบ ซึ่งแสดงค่าω (omega) โดย SYS คือ Transfer Function ของระบบ

Function PHASE(SYS)

ผลการทำงานของฟังก์ชันจะคำนวนหาค่า Phase ของ Transfer Function ของระบบ ซึ่งแสดงค่าω (omega) โดย SYS คือ Transfer Function ของระบบ

Function POLY2COF(POLY,VAR)

ผลการทำงานของฟังก์ชันจะคำนวนหาค่าสัมประสิทธิ์ของพหุนาม โดย POLY คือพหุนาม, VAR คือ ตัวแปรของพหุนาม

Function ROOTS(COEF)

ผลการทำงานของฟังก์ชันจะคำนวนหาคำตอบของพหุนาม จากสัมประสิทธิ์ของพหุนาม โดย COEF คือ สัมประสิทธิ์ของพหุนาม

Function ROUTH(POLY,VAR)

ผลการทำงานของฟังก์ชันจะสร้างตาราง Routh-Hurwitz เป็นเมตริกซ์ โดย POLY คือ พหุนาม, VAR คือ ตัวแปรต้นของพหุนาม

Function INVLAP(SYS)

ผลการทำงานของฟังก์ชันจะคำนวนหาค่าInverse Laplace จาก S-Domain เป็น Time-domain ในตัวแปร t

Function PID(SYS)

ผลการทำงานของฟังก์ชันจะคำนวนหาค่า Kp, Ti, Td ในการปรับค่า Controller ซึ่งแบ่งเป็นสามแบบ คือ

1. P-controller
2. PI-controller
3. PID-controller

แล้วใช้หลักการจูนพีไอดี(PID) Ziegler-Nichols ในการหาค่า Kp, Ti, Td

Function CLLOOP(Ns,Ds)

ผลการทำงานของฟังก์ชันจะคำนวนหา Close Loop ของ Transfer Function ของระบบ Y(S)/U(S)

ผลการทดลองการทำงานของโปรแกรม JKI Toolbox ในการปรับแต่งระบบควบคุมแบบพีไอดี (PID)

ในการทดสอบโปรแกรม JKI Toolbox ในการปรับแต่งระบบควบคุมแบบพีไอดี (PID) จะเป็นการคำนวณหาค่า Kp, Ti, Td ระหว่างการคำนวณโดยใช้โปรแกรม JKI Toolbox เปรียบเทียบกับการคำนวณโดยวิธีของ Ziegler-Nichols จะได้ผลการทดลองดังนี้

จากการทดลองการทำงานของโปรแกรมเราได้เลือกระบบในทางอุตสาหกรรมที่มีทรานสเฟอร์ฟังก์ชันดังนี้

$$G(s) = \frac{1}{s(s+1)(s+5)}$$

ผลการทดลองการคำนวณหาค่า Kp, Ti, Td โดยใช้ฟังก์ชัน PID(SYS) ของโปรแกรม JKI Toolbox จากการทดลองสามารถหาค่า Kp, Ti, Td ดังนี้

EX : jki\PID(1/(s(s+1)/(s+5)))

ANSWER :

PID-Controller : Kp = 18 Ti = 1.4 Td = 0.351

รูปที่ 5.1.1แสดง Function PID

แสดงผลการทดลองการคำนวณหาค่า Kp, Ti, Td โดยใช้วิธี Ziegler-Nichols Rules

การปรับแต่งระบบควบคุมแบบพีไอดี โดยใช้วิธี Ziegler-Nichols Rules จะได้ PID Controller ของระบบมีฟังก์ชันเป็น

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$

จากอินทิเกรเตอร์ของระบบจะได้ทรานสเฟอร์ฟังก์ชันของระบบเป็น

$$\frac{C(s)}{R(s)} = \frac{K_p}{s(s+1)(s+5)+K_p}$$

คำนวณหาค่า Kp ที่ทำให้ระบบมีเสถียรภาพจากการใช้ Routh table จากสมการคุณลักษณะของระบบคือ

$$s^3 + 6s^2 + 5s + K_p = 0$$

จะได้ Routh array เป็น

s^3	1	5
s^2	6	K_p
s^1	$\frac{30-K_p}{6}$	
s^0	K_p	

จาก Routh array เราเลือกค่า Kp=30 ดังนั้น Criticle Gain Kcr จะมีค่าเป็น

$$K_{cr} = 30$$

จากการกำหนดค่าเกน Kp ให้มีค่าเท่ากับ Kcr (=30) จะได้สมการคุณลักษณะเป็น

$$s^3 + 6s^2 + 5s + 30 = 0$$

คำนวณหา Frequency of sustained oscillation โดยกำหนดให้ s=jω จะได้สมการคุณลักษณะเป็น

$$(j\omega)^3 + 3(j\omega)^2 + 5(j\omega) + 30 = 0$$

จัดรูปสมการจะได้

$$6(5 - \omega^2) + j\omega(5 - \omega^2) = 0$$

จะได้

$$\omega^2 = 5$$

$$\omega = \sqrt{5}$$

ดังนั้น period of sustained oscillation จะมีค่าเป็น

$$P_{cr} = \frac{2\pi}{\omega} = \frac{2\pi}{\sqrt{5}} = 2.8099$$

จะสามารถหาค่า Kp, Ti, Td ได้เป็น

$$K_p = 0.6K_{cr} = 18$$

$$T_i = 0.5P_{cr} = 1.405$$

$$T_d = 0.125P_{cr} = 0.35124$$

จากผลการทดลองนี้ ค่า Kp, Ti, Td ของระบบควบคุมพีไอดีที่คำนวณได้จากทั้งสองวิธีมีค่าตรงกัน

5.2 ตัวอย่างการนำ JKI Toolbox ไปประยุกต์ใช้งานจริง

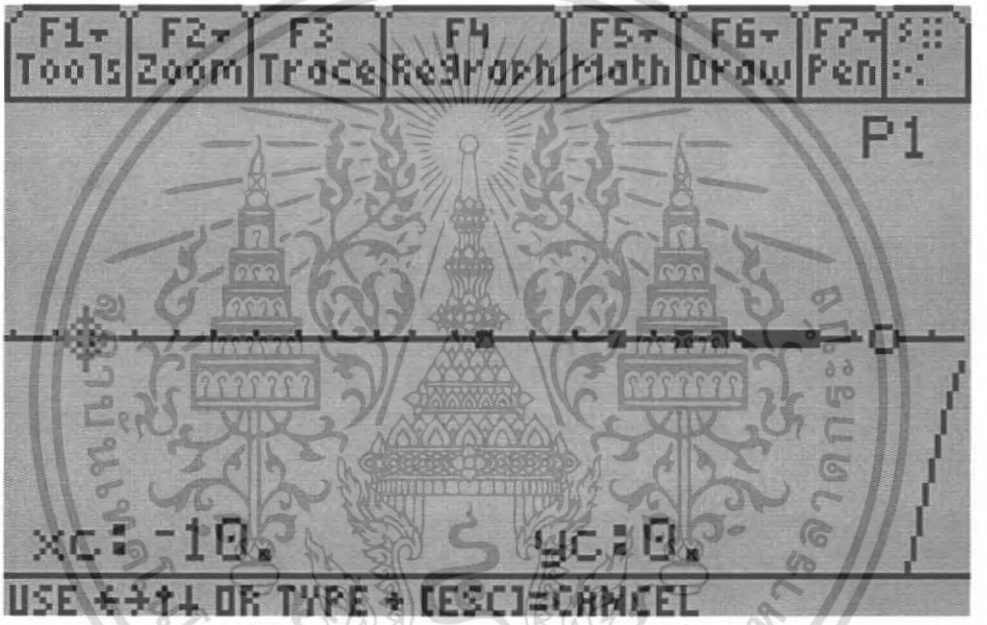
JKI Toolbox เป็นโปรแกรมช่วยในการคำนวณทางคณิตศาสตร์ของระบบควบคุม และการคำนวณทางคณิตศาสตร์สำหรับการปรับแต่งระบบควบคุมแบบพีไอดี (PID) โดยการใช้งานโปรแกรม JKI Toolbox สามารถใช้งานผ่านทางเครื่องคิดเลขรุ่นTI-89โดยการดาวน์โหลดโปรแกรม JKI Toolbox ลงในตัวเครื่องคิดเลขจากเครื่องคอมพิวเตอร์โดยภายในตัวโปรแกรมจะมีฟังก์ชันการใช้งานในการคำนวณเฉพาะอย่างทางคณิตศาสตร์ของระบบควบคุม และจะแยกเป็นฟังก์ชันการใช้งานแตกต่างกันออกไป เช่น ฟังก์ชันการคำนวณหาค่าโพล และค่าซีโร่ของระบบควบคุม ฟังก์ชันการแสดงกราฟของตำแหน่งโพล และซีโร่ ของระบบควบคุม ฟังก์ชันการคำนวณหาคำตอบของอินเวอส์ลาปลาส(Inverse Laplace)ฟังก์ชันการหาค่าของ Kp Ti Td และการปรับแต่งค่าของระบบควบคุมแบบพีไอดี (PID) ฟังก์ชันการหาทรานสเฟอร์ฟังก์ชัน (Transfer Function) ของระบบ ฟังก์ชันการคำนวณตารางเราธ์-เฮอร์วิธ โดยสามารถเรียกใช้งานฟังก์ชันการคำนวณต่างๆเหล่านี้ผ่านทางฟังก์ชันเมนูของโปรแกรม JKI Toolbox ซึ่งฟังก์ชันต่างๆ ของโปรแกรม JKI Toolbox จะถูกเก็บอยู่ใน Folder JKI และอยู่ในสภาพพร้อมใช้งานโดยโปรแกรม JKI Toolbox มีวิธีใช้ดังตัวอย่างการนำโปรแกรม JKI Toolbox ไปใช้งานจริงด้านวิศวกรรมระบบควบคุม

- การนำโปรแกรม JKI Toolbox คำนวณหา Root Locus ของระบบในอุตสาหกรรมที่มีทรานสเฟอร์ฟังก์ชันเป็น

$$G(s) = \frac{s+1}{s+10}$$

EX: jki\RLOCUS((s+1)/(s+10))

แสดงผลการคำนวณหา Root Locus ของระบบควบคุม



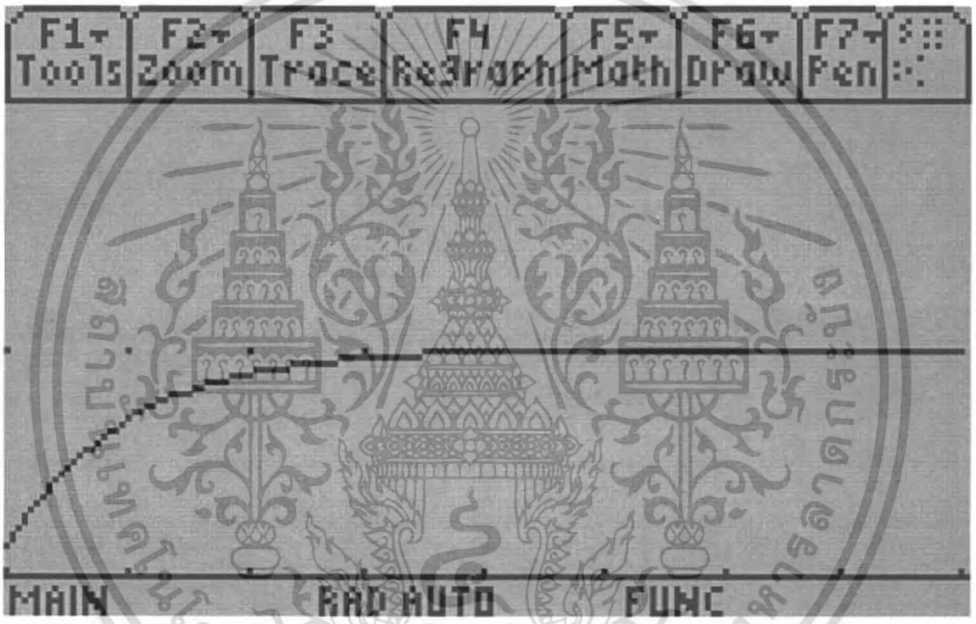
รูปที่ 5.2.1 แสดง ผลการทำงานของFunction RLOCUS

- การนำโปรแกรม JKI Toolbox คำนวณหา Step Response ของระบบในอุตสาหกรรมที่มีทรานสเฟอร์ฟังก์ชันเป็น

$$G(s) = \frac{1}{s+1}$$

EX: : jki\STEP(1/(s+1))

แสดงผลการคำนวณหา Step Response ของระบบควบคุม



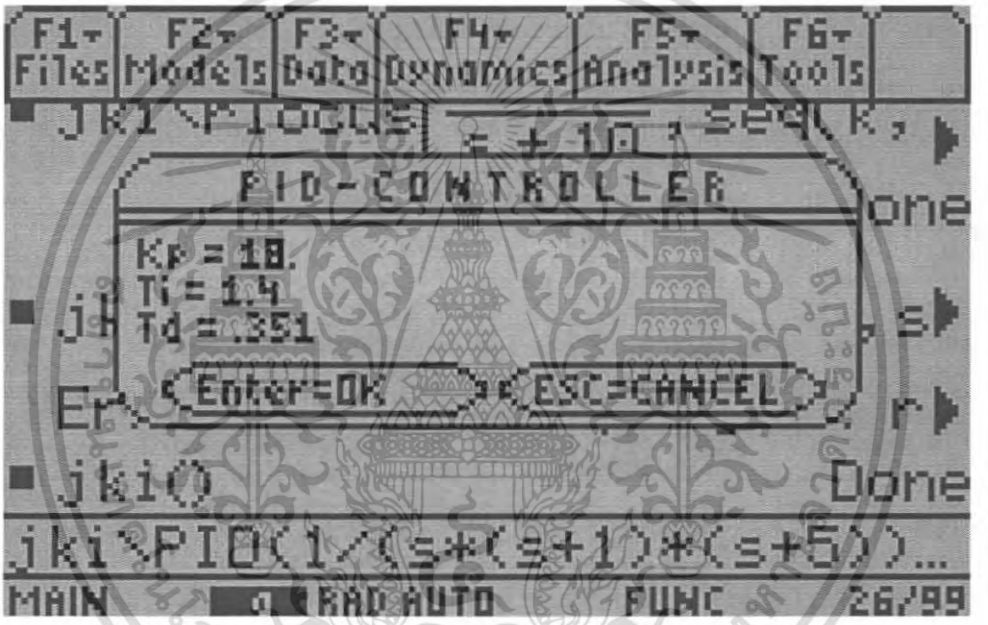
รูปที่ 5.2.2 แสดงผลการทำงานของFunction PHASE

- การนำโปรแกรม JKI Toolbox ในการปรับแต่งระบบควบคุมแบบพีไอดี (PID) ของระบบในอุตสาหกรรมที่มีทรานสเฟอร์ฟังก์ชันเป็น

$$G(s) = \frac{1}{s(s+1)(s+5)}$$

EX: : jki\PID(1/(s(s+1)(s+5)))

แสดงผลการคำนวณ เพื่อปรับแต่งค่าของระบบควบคุมแบบพีไอดี (PID)



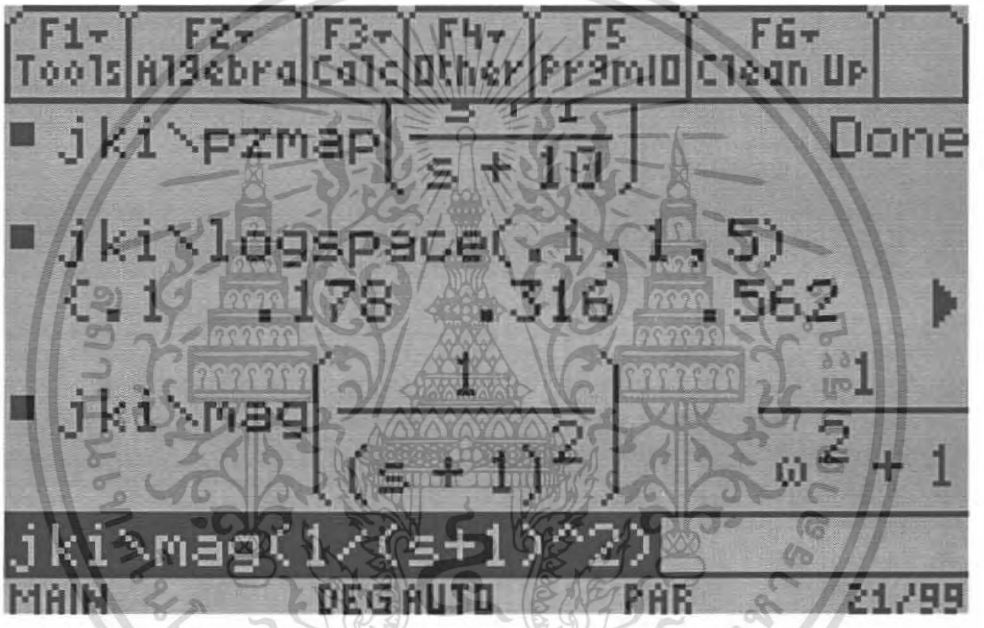
รูปที่ 5.2.3 แสดงผลการทำงานของFunction PID

- การนำโปรแกรม JKI Toolbox คำนวณหา Magnitude ของระบบในอุตสาหกรรมที่มีทรานสเฟอร์ฟังก์ชันเป็น

$$G(s) = \frac{1}{(s+1)^2}$$

EX: jki\MAG(1/(s+1)^2)

แสดงผลการคำนวณหาค่า Magnitude ของระบบ



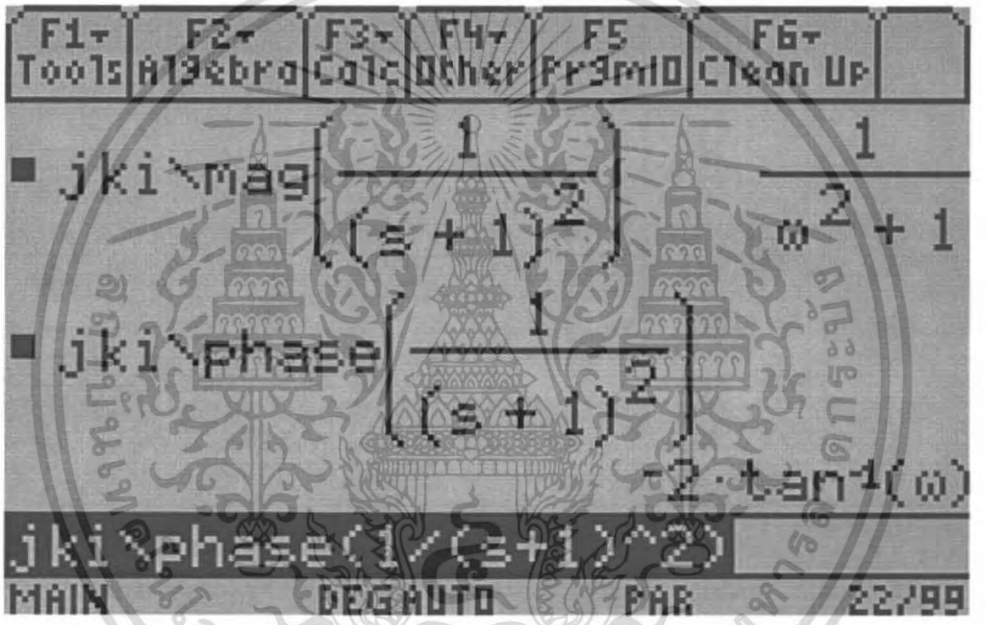
รูปที่ 5.2.4 แสดงผลการทำงานของFunction MAG

- การนำโปรแกรม JKI Toolbox คำนวณหา Phase ของระบบในอุตสาหกรรมที่มีทรานสเฟอร์ฟังก์ชันเป็น

$$G(s) = \frac{1}{(s+1)^2}$$

EX: jki\PHASE(1/(s+1)^2)

แสดงผลการคำนวณหาค่า Phase ของระบบ



รูปที่ 5.2.5 แสดงผลการทำงานของFunction PHASE

บทที่ 6
บทสรุปและบทวิจารณ์

6.1 สรุปผลการทดลอง

จากการทดสอบการทำงานของโปรแกรม JKI Toolbox พบว่าฟังก์ชันการใช้งานของโปรแกรมสามารถใช้งานในการคำนวณทางคณิตศาสตร์ในเชิงระบบควบคุมได้โดยสามารถให้ผลลัพธ์ในเชิงระบบควบคุมที่ต้องการนำไปใช้งานทางวิศวกรรมระบบควบคุมโดยแต่ละฟังก์ชันการใช้งานของโปรแกรมJKI Toolbox สามารถแสดงผลลัพธ์ในรูปแบบต่างๆของผลตอบสนองทางระบบควบคุมได้และค่าผลลัพธ์ที่คำนวณได้ตรงกับผลลัพธ์จากการคำนวนโดยใช้ฟังก์ชันการทำงานของโปรแกรม MATLAB ซึ้งในขณะนี้โปรแกรม JKI Toolbox มีฟังก์ชันการคำนวณทางด้านวิศวกรรมระบบควบคุมและการปรับแต่งระบบควบคุม PID แต่ในอนาคต โปรแกรมจะสามารถพัฒนา เพื่อไปทำงานในด้านการคำนวณในสาขาอื่นๆได้อีก

6.2 บทวิจารณ์

การพัฒนาด้านโปรแกรมนี้เป็นการเลียนแบบการทำงานของโปรแกรม MATLAB ซึ่งมีการออกแบบฟังก์ชันการทำงานของโปรแกรม JKI Toolbox ให้สอดคล้องกับการทำงานของฟังก์ชันการทำงานของโปรแกรม MATLAB และสามารถนำโปรแกรมนี้ไปใช้งานผ่านเครื่องคำนวณที่มีความสามารถสูงได้

เนื่องจากฟังก์ชันการทำงานของโปรแกรม JKI Toolbox ถูกสร้างขึ้นเลียนแบบการทำงานจากฟังก์ชันการทำงานของโปรแกรม MATHLAB ทำให้ค่าที่คำนวณได้จากฟังก์ชันการทำงานของโปรแกรมทั้งสองมีค่าตรงกัน

จากการทดสอบการทำงานของตัวโปรแกรม พบว่า ตัวโปรแกรม JKI Toolbox สามารถนำไปคำนวณค่าในการปรับแต่งระบบควบคุมแบบ PID ได้โดยให้ค่าที่คำนวณด้วยโปรแกรม MATLAB และสามารถใช้งานโปรแกรม JKI Toolbox ผ่านทางเครื่องคำนวณรุ่น TI-89 ได้ ทำให้สามารถนำไปใช้งานได้จริงอย่างมีประสิทธิภาพ

6.3 ปัญหาที่พบและแนวทางในการพัฒนา

ในการออกแบบตัวโปรแกรมและทดสอบการใช้งานได้จริงของโปรแกรม JKI Toolbox มีดังนี้

- การศึกษาเครื่องคำนวณ (Calculator) รุ่น TI-89 ต้องใช้วเวลาในการศึษานานพอสมควร เนื่องมาจากนักศึกษาไม่เคยได้ใช้งานเครื่องคำนวณรุ่นนี้มาก่อน ทำให้ใช้เวลาในการศึกษานาน
- การทดสอบการทำงานของโปรแกรม JKI Toolbox ในเครื่องคำนวณเกิดปัญหาการดาวน์โหลดโปรแกรมอาจผิดพลาดเพราะตัวเครื่องคำนวณที่ใช้ทดสอบนั้นเป็นเพียงโปรแกรมของเครื่องคำนวณรุ่น TI-89
- การทำงานของเครื่องคำนวณรุ่น TI-89 อาจเกิดการผิดพลาดเนื่องจากเป็นโปรแกรมเครื่องคำนวณจำลองและทดสอบผ่านโปรแกรมเครื่องคำนวณจำลอง จึงทำให้เกิดความผิดพลาด (ERROR) จากโปรแกรม

แนวทางการแก้ไข

- แบ่งงานกันทำในแต่ละส่วนเพื่อให้เกิดการทำงานที่มีประสิทธิภาพ และความรวดเร็วในการทำงาน
- ลงทุนซื้อเครื่องคำนวณรุ่น TI-89 มาทดสอบการใช้งานของตัวโปรแกรม เพื่อป้องกันความผิดพลาด ที่เกิดขึ้นจากโปรแกรมเครื่องคำนวณจำลอง
- เพิ่มจำนวนผู้ปฏิบัติงานเพื่อเพิ่มประสิทธิภาพการทำงานได้อย่างเต็มที่

แนวทางในการค้นคว้าและพัฒนา

ในส่วนของการทำงานของตัวโปรแกรม JKI Toolbox สามารถใช้งานในการคำนวณทางด้านระบบควบคุม และการปรับแต่งระบบควบคุมแบบ PID ได้ และตัวโปรแกรมนี้สามารถออกแบบให้รองรับการใช้งานทางด้านการคำนวณทางคณิตศาสตร์ในสาขาต่างๆได้ ทำให้ในอนาคต อาจจะสามารถพัฒนาและออกแบบตัวโปรแกรมให้สามารถครอบคลุมการทำงานทางด้านการคำนวณทางคณิตศาสตร์ทั้งหมดได้

เอกสารอ้างอิง

[1] Kutsuhiko Ogata.Modern Control Engineering.London:Prentice-Hall International,c1990.

[2] Frank L Lewis.Applied Optimal Control & Estimation.Englewood cliffs,N.J.:Prentice-Hall,c1992.

[3] Baosar,Tamer.Control Theory.New York:IEEE Press,c2001.

[4] Vladimir Zakian.Automatic Control.New York:Springle Sciences,c2005

[5] Ivan Graham."**MATLAB Manual and Introductory Tutorials Mathematical Sciencess, University of Bath.**" [Online].Available: http://www.sonoma.edu/user/k/kujoory/course_materials/ces_400/a_manual-matlab.pdf

[6] A.Anton and Robert C. Busty. **"Contemporary Linear Algebra:Ti89 calculator Technology Resource Manual."** [Online].Available: http://www.dealtime.com/xGS-Ti_89_Manual~NS-1~linkin_id-8009364

ภาคผนวก

หลักการและวิธีการจูนพีไอดี

(Kutsuhiko Ogata.Modern Control Engineering.London:Prentice-Hall International,c1990.)

10

PID Controls and Introduction to Robust Control

10–1 INTRODUCTION

In previous chapters we occasionally discussed the basic PID control schemes. For example, in Chapter 5 we presented hydraulic, pneumatic, and electronic PID controllers. In Chapters 7 and 9 we designed control systems where PID controllers were involved.

In this chapter we first present tuning rules for the basic PID controllers and then discuss modified forms of PID control schemes, including PI-D control, I-PD control, and two-degrees-of-freedom PID control. Finally, we introduce the concept of robust design.

It is interesting to note that more than half of the industrial controllers in use today utilize PID or modified PID control schemes. Analog PID controllers are mostly hydraulic, pneumatic, electric, and electronic types or their combinations. Currently, many of these are transformed into digital forms through the use of microprocessors.

Because most PID controllers are adjusted on site, many different types of tuning rules have been proposed in the literature. Using these tuning rules, delicate and fine tuning of PID controllers can be made on site. Also, automatic tuning methods have been developed and some of the PID controllers may possess on-line automatic tuning capabilities. Modified forms of PID control, such as I-PD control and two-degrees-of-freedom PID control, are currently in use in industry. Many practical methods for bumpless switching (from manual operation to automatic operation) and gain scheduling are commercially available.

The usefulness of PID controls lies in their general applicability to most control systems. In the field of process control systems, it is a well-known fact that the basic and

modified PID control schemes have proved their usefulness in providing satisfactory control, although they may not provide optimal control in many given situations.

Outline of the chapter. Section 10–1 has presented introductory material for the chapter. Section 10–2 deals with tuning methods for the basic PID control, commonly known as Ziegler–Nichols tuning rules. Section 10–3 discusses modified PID control schemes, such as PI-D control and I-PD control. Section 10–4 introduces two-degrees-of-freedom PID control schemes. Section 10–5 introduces the concept of robust control using a two-degrees-of-freedom control system as an example.

10–2 TUNING RULES FOR PID CONTROLLERS

PID control of plants. Figure 10–1 shows a PID control of a plant. If a mathematical model of the plant can be derived, then it is possible to apply various design techniques for determining parameters of the controller that will meet the transient and steady-state specifications of the closed-loop system. However, if the plant is so complicated that its mathematical model cannot be easily obtained, then analytical approach to the design of a PID controller is not possible. Then we must resort to experimental approaches to the tuning of PID controllers.

The process of selecting the controller parameters to meet given performance specifications is known as controller tuning. Ziegler and Nichols suggested rules for tuning PID controllers (meaning to set values K_p, T_i, and T_d) based on experimental step responses or based on the value of K_p that results in marginal stability when only the proportional control action is used. Ziegler–Nichols rules, which are presented in the following, are very convenient when mathematical models of plants are not known. (These rules can, of course, be applied to the design of systems with known mathematical models.)

Ziegler–Nichols rules for tuning PID controllers. Ziegler and Nichols proposed rules for determining values of the proportional gain K_p, integral time T_i, and derivative time T_d based on the transient response characteristics of a given plant. Such determination of the parameters of PID controllers or tuning of PID controllers can be made by engineers on site by experiments on the plant. (Numerous tuning rules for PID controllers have been proposed since the Ziegler–Nichols proposal. They are available in the literature. Here, however, we introduce only the Ziegler–Nichols tuning rules.)

There are two methods called Ziegler–Nichols tuning rules. In both methods, they aimed at obtaining 25% maximum overshoot in step response (see Figure 10–2).

First method. In the first method, we obtain experimentally the response of the plant to a unit-step input, as shown in Figure 10–3. If the plant involves neither inte-

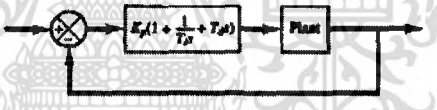


Figure 10–1
PID control of a plant.

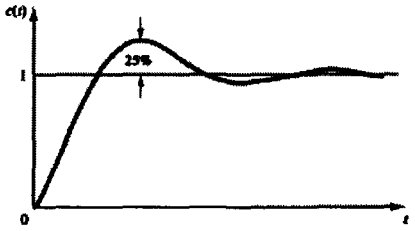


Figure 10–2
Unit-step response curve showing 25% maximum overshoot.

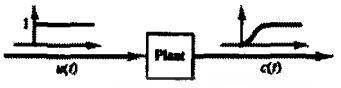


Figure 10–3
Unit-step response of a plant.

grator(s) nor dominant complex-conjugate poles, then such a unit-step response curve may look like an S-shaped curve, as shown in Figure 10–4. (If the response does not exhibit an S-shaped curve, this method does not apply.) Such step-response curves may be generated experimentally or from a dynamic simulation of the plant.

The S-shaped curve may be characterized by two constants, delay time L and time constant T. The delay time and time constant are determined by drawing a tangent line at the inflection point of the S-shaped curve and determining the intersections of the tangent line with the time axis and line $c(t) = K$, as shown in Figure 10–4. The transfer function $C(s)/U(s)$ may then be approximated by a first-order system with a transport lag as follows:

$$\frac{C(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts + 1}$$

Ziegler and Nichols suggested to set the values of K_p, T_i, and T_d according to the formula shown in Table 10–1.

Notice that the PID controller tuned by the first method of Ziegler–Nichols rules gives

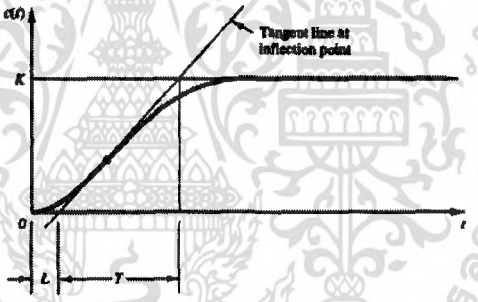


Figure 10–4
S-shaped response curve.

Table 10–1 Ziegler-Nichols Tuning Rule Based on Step Response of Plant (First Method)

Type of Controller	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$
$$= 1.2\frac{T}{L}\left(1 + \frac{1}{2Ls} + 0.5Ls\right)$$
$$= 0.6T\frac{\left(s + \frac{1}{L}\right)^2}{s}$$

Thus, the PID controller has a pole at the origin and double zeros at $s = -1/L$.

Second method. In the second method, we first set $T_i = \infty$ and $T_d = 0$. Using the proportional control action only (see Figure 10–5), increase K_p from 0 to a critical value K_{cr} where the output first exhibits sustained oscillations. (If the output does not exhibit sustained oscillations for whatever value K_p may take, then this method does not apply.) Thus, the critical gain K_{cr} and the corresponding period P_{cr} are experimentally determined (see Figure 10–6). Ziegler and Nichols suggested that we set the values of the parameters K_p, T_i, and T_d according to the formula shown in Table 10–2.

Figure 10–5
Closed-loop system with a proportional controller.

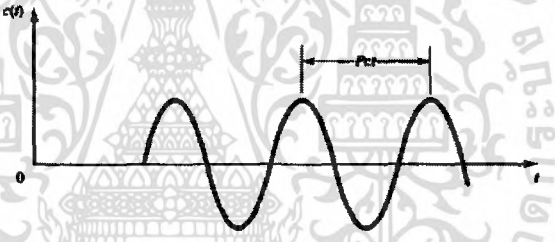


Figure 10–6
Sustained oscillation with period P_{cr}.

Table 10–2 Ziegler–Nichols Tuning Rule Based on Critical Gain K_{cr} and Critical Period P_{cr} (Second Method)

Type of Controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Notice that the PID controller tuned by the second method of Ziegler–Nichols rules gives

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$
$$= 0.6K_{cr}\left(1 + \frac{1}{0.5P_{cr}s} + 0.125\,P_{cr}s\right)$$
$$= 0.075K_{cr}P_{cr}\frac{\left(s + \frac{4}{P_{cr}}\right)^2}{s}$$

Thus, the PID controller has a pole at the origin and double zeros at $s = -4/P_{cr}$.

Comments. Ziegler–Nichols tuning rules (and other tuning rules presented in the literature) have been widely used to tune PID controllers in process control systems where the plant dynamics are not precisely known. Over many years, such tuning rules proved to be very useful. Ziegler–Nichols tuning rules can, of course, be applied to plants whose dynamics are known. (If plant dynamics are known, many analytical and graphical approaches to the design of PID controllers are available, in addition to Ziegler–Nichols tuning rules.)

If the transfer function of the plant is known, a unit-step response may be calculated or the critical gain K_{cr} and critical period P_{cr} may be calculated. Then, using those calculated values, it is possible to determine the parameters K_p, T_i, and T_d from Table 10–1 or 10–2. However, the real usefulness of Ziegler–Nichols tuning rules (and other tuning rules) becomes apparent when the plant dynamics are not known so that no analytical or graphical approaches to the design of controllers are available.

Generally, for plants with complicated dynamics but no integrators, Ziegler–Nichols tuning rules can be applied. However, if the plant has an integrator, these rules may not be applied in some cases. To illustrate such a case where Ziegler–Nichols rules do not apply, consider the following case: Suppose that a unity-feedback control system has a plant whose transfer function is

$$G(s) = \frac{(s + 2)(s + 3)}{s(s + 1)(s + 5)}$$

Because of the presence of an integrator, the first method does not apply. Referring to

Figure 10–3, the step response of this plant will not have an S-shaped response curve; rather, the response increases with time. Also, if the second method is attempted (see Figure 10–5), the closed-loop system with a proportional controller will not exhibit sustained oscillations whatever value the gain K_p may take. This can be seen from the following analysis. Since the characteristic equation is

$$s(s + 1)(s + 5) + K_p(s + 2)(s + 3) = 0$$

or

$$s^3 + (6 + K_p)s^2 + (5 + 5K_p)s + 6K_p = 0$$

the Routh array becomes

$$\begin{array}{lll} s^3 & 1 & 5 + 5K_p \\ s^2 & 6 + K_p & 6K_p \\ s^1 & \dfrac{30 + 29K_p + 5K_p^2}{6 + K_p} & 0 \\ s^0 & 6K_p & \end{array}$$

The coefficients in the first column are positive for all values of positive K_p. Thus, in the present case the closed-loop system will not exhibit sustained oscillations and, therefore, the critical gain value K_{cr} does not exist. Hence, the second method does not apply.

If the plant is such that Ziegler–Nichols rules can be applied, then the plant with a PID controller tuned by Ziegler–Nichols rules will exhibit approximately 10% ~ 60% maximum overshoot in step response. On the average (experimented on many different plants), the maximum overshoot is approximately 25%. (This is quite understandable because the values suggested in Tables 10–1 and 10–2 are based on the average.) In a given case, if the maximum overshoot is excessive, it is always possible (experimentally or otherwise) to make fine tuning so that the closed-loop system will exhibit satisfactory transient responses. In fact, Ziegler–Nichols tuning rules give an educated guess for the parameter values and provide a starting point for fine tuning.

EXAMPLE 10–1 Consider the control system shown in Figure 10–7 in which a PID controller is used to control the system. The PID controller has the transfer function

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$

Although many analytical methods are available for the design of a PID controller for the present system, let us apply a Ziegler–Nichols tuning rule for the determination of the values of parameters K_p, T_i, and T_d. Then obtain a unit-step response curve and check to see if the designed system exhibits approximately 25% maximum overshoot. If the maximum overshoot is excessive (40% or more), make a fine tuning and reduce the amount of the maximum overshoot to approximately 25%.

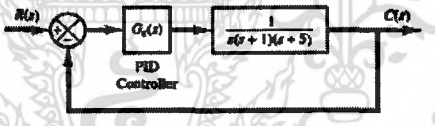


Figure 10–7 PID-controlled system.

Since the plant has an integrator, we use the second method of Ziegler–Nichols tuning rules. By setting $T_i = \infty$ and $T_d = 0$, we obtain the closed-loop transfer function as follows:

$$\frac{C(s)}{R(s)} = \frac{K_p}{s(s+1)(s+5) + K_p}$$

The value of K_p that makes the system marginally stable so that sustained oscillation occurs can be obtained by use of Routh's stability criterion. Since the characteristic equation for the closed-loop system is

$$s^3 + 6s^2 + 5s + K_p = 0$$

the Routh array becomes as follows:

s^3	1	5
s^2	6	K_p
s^1	$\frac{30 - K_p}{6}$	
s^0	K_p	

Examining the coefficients of the first column of the Routh table, we find that sustained oscillation will occur if $K_p = 30$. Thus, the critical gain K_{cr} is

$$K_{cr} = 30$$

With gain K_p set equal to K_{cr} (= 30), the characteristic equation becomes

$$s^3 + 6s^2 + 5s + 30 = 0$$

To find the frequency of the sustained oscillation, we substitute $s = j\omega$ into this characteristic equation as follows:

$$(j\omega)^3 + 6(j\omega)^2 + 5(j\omega) + 30 = 0$$

or

$$6(5 - \omega^2) + j\omega(5 - \omega^2) = 0$$

from which we find the frequency of the sustained oscillation to be $\omega^2 = 5$ or $\omega = \sqrt{5}$. Hence, the period of sustained oscillation is

$$P_{cr} = \frac{2\pi}{\omega} = \frac{2\pi}{\sqrt{5}} = 2.8099$$

Referring to Table 10–2, we determine K_p, T_i, and T_d as follows:

$$K_p = 0.6K_{cr} = 18$$
$$T_i = 0.5P_{cr} = 1.405$$
$$T_d = 0.125P_{cr} = 0.35124$$

The transfer function of the PID controller is thus

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$
$$= 18\left(1 + \frac{1}{1.405s} + 0.35124s\right)$$
$$= \frac{6.3223(s + 1.4235)^2}{s}$$

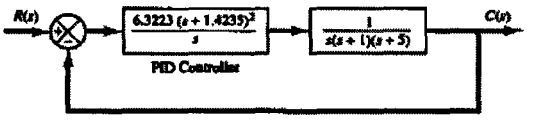


Figure 10–8
Block diagram of the system with PID controller designed by use of Ziegler–Nichols tuning rule (second method).

The PID controller has a pole at the origin and double zero at $s = -1.4235$. A block diagram of the control system with the designed PID controller is shown in Figure 10–8.

Next, let us examine the unit-step response of the system. The closed-loop transfer function $C(s)/R(s)$ is given by

$$\frac{C(s)}{R(s)} = \frac{6.3223s^2 + 18s + 12.811}{s^4 + 6s^3 + 11.3223s^2 + 18s + 12.811}$$

The unit-step response of this system can be obtained easily with MATLAB. See MATLAB Program 10–1. The resulting unit-step response curve is shown in Figure 10–9. The maximum overshoot in the unit-step response is approximately 62%. The amount of maximum overshoot is excessive. It can be reduced by fine tuning the controller parameters. Such fine tuning can be made on the computer. We find that by keeping $K_p = 18$ and by moving the double zero of the PID controller to $s = -0.65$, that is, using the PID controller

$$G_c(s) = 18\left(1 + \frac{1}{3.077s} + 0.7692s\right) = 13.846\,\frac{(s + 0.65)^2}{s} \qquad (10\text{–}1)$$

the maximum overshoot in the unit-step response can be reduced to approximately 18% (see Figure 10–10). If the proportional gain K_p is increased to 39.42, without changing the location of the double zero ($s = -0.65$), that is, using the PID controller

$$G_c(s) = 39.42\left(1 + \frac{1}{3.077s} + 0.7692s\right) = 30.322\,\frac{(s + 0.65)^2}{s} \qquad (10\text{–}2)$$

then the speed of response is increased, but the maximum overshoot is also increased to approximately 28%, as shown in Figure 10–11. Since the maximum overshoot in this case is fairly close to 25% and the response is faster than the system with $G_c(s)$ given by Equation (10–1), we may consider $G_c(s)$ as given by Equation (10–2) as acceptable. Then the tuned values of K_p, T_i, and T_d become

$$K_p = 39.42, \qquad T_i = 3.077, \qquad T_d = 0.7692$$

It is interesting to observe that these values respectively are approximately twice the values suggested by the second method of the Ziegler–Nichols tuning rule. The important thing to note here is that the Ziegler–Nichols tuning rule has provided a starting point for fine tuning.

It is instructive to note that, for the case where the double zero is located at $s = -1.4235$, increasing the value of K_p increases the speed of response, but as far as the percentage maximum

MATLAB Program 10–1

```
% ---------- Unit-step response ----------

num = [0   0   6.3223   18   12.811];
den = [1   6   11.3223   18   12.811];
step(num,den)
grid
title('Unit-Step Response')
```

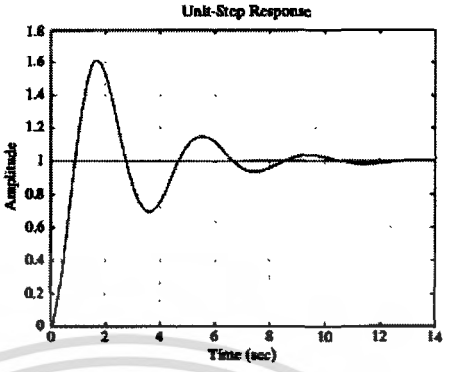


Figure 10–9
Unit-step response curve of PID-controlled system designed by use of Ziegler–Nichols tuning rule (second method).

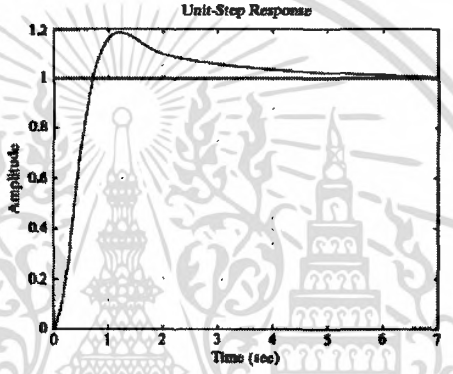


Figure 10–10
Unit-step response of the system shown in Figure 10–7 with PID controller having parameters $K_p = 18$, $T_i = 3.077$, and $T_d = 0.7692$.

overshoot is concerned, varying gain K_p has very little effect. The reason for this may be seen from the root-locus analysis. Figure 10–12 shows the root-locus diagram for the system designed by use of the second method of Ziegler–Nichols tuning rules. Since the dominant branches of root loci are along the $\zeta = 0.3$ lines for a considerable range of K, varying the value of K (from 6 to 30) will not change the damping ratio of the dominant closed-loop poles very much. However, varying the location of the double zero has a significant effect on the maximum overshoot, because the damping ratio of the dominant closed-loop poles can be changed significantly. This can also be seen from the root-locus analysis. Figure 10–13 shows the root-locus diagram for the system where the PID controller has the double zero at $s = -0.65$. Notice the change of the root-locus configuration. This change in the configuration makes it possible to change the damping ratio of the dominant closed-loop poles.

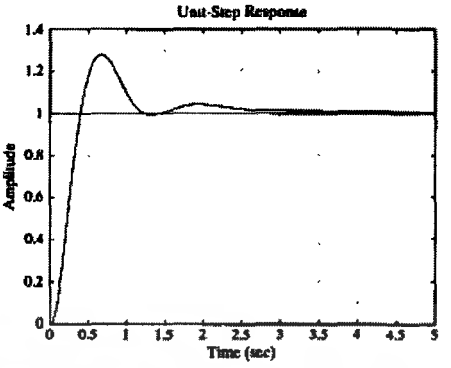


Figure 10–11
Unit-step response of the system shown in Figure 10–7 with PID controller having parameters $K_p = 39.42$, $T_i = 3.077$, and $T_d = 0.7692$.

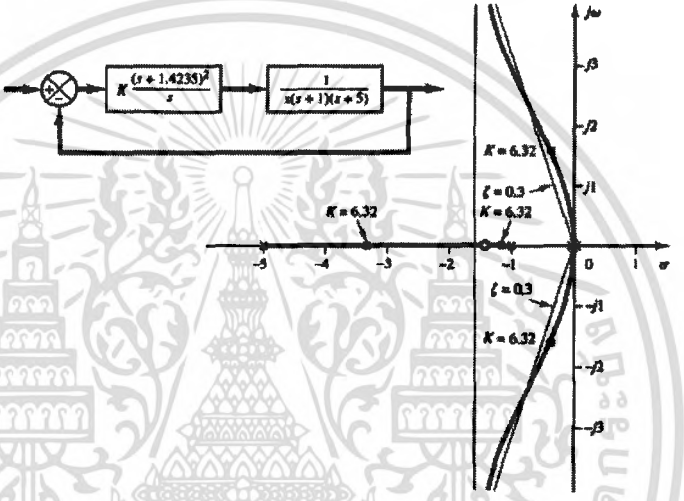


Figure 10–12
Root-locus diagram of system when PID controller has double zero at $s = -1.4235$.

In Figure 10–13, notice that, in the case where the system has gain $K = 30.322$, the closed-loop poles at $s = -2.35 \pm j4.82$ act as dominant poles. Two additional closed-loop poles are very near the double zero at $s = -0.65$, with the result that these closed-loop poles and the double zero almost cancel each other. The dominant pair of closed-loop poles indeed determines the nature of the response. On the other hand, when the system has $K = 13.846$, the closed-loop poles at $s = -2.35 \pm j2.62$ are not quite dominant because the two other closed-loop poles near the dou-

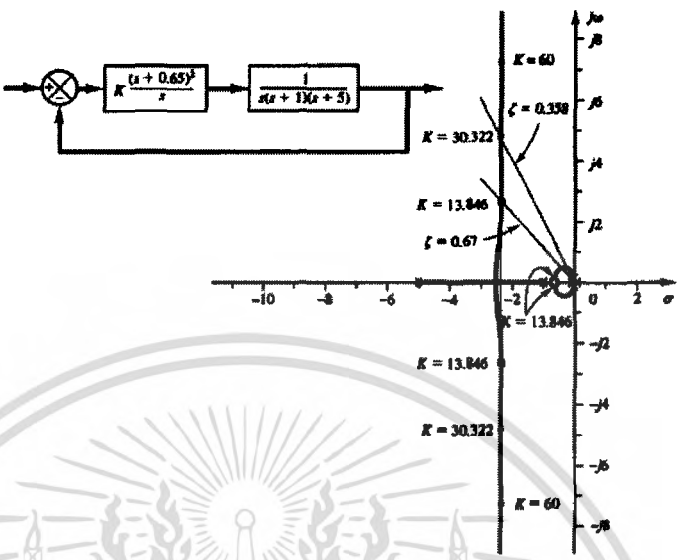


Figure 10–13
Root-locus diagram of system when PID controller has double zero at $s = -0.65$. $K = 13.846$ corresponds to $G_c(s)$ given by Equation (10–1) and $K = 30.322$ corresponds to $G_c(s)$ given by Equation (10–2).

ble zero at $s = -0.65$ have considerable effect on the response. The maximum overshoot in the step response in this case (18%) is much larger than the case where the system is of second-order having only dominant closed-loop poles. (In the latter case the maximum overshoot in the step response would be approximately 6%.)