บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
แบบฟอร์มการให้คะแนนการสอบวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การลดทอนแบบในระบบเชิงเส้น
(Model Reduction in Linear Systems)"

ชื่อนักศึกษา นายสัญญา สายะสถิตย์ **รหัสประจำตัว** 31126-022
หลักสูตร วิศวกรรมศาสตรมหาบัณฑิต **สาขาวิชา** วิศวกรรมไฟฟ้า
ภาควิชา ระบบควบคุม
อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.วิพันธ์ ปรีชาพานิช

ชื่ออาจารย์ผู้ควบคุมการสอบ	ลายมือชื่อ
รศ.วิพันธ์ ปรีชาพานิช	
รศ.ดร.โกศล เพ็ชร์สุวรรณ์	
ผศ.ดร.จงกล งามวิวิทย์	
ผศ.ดร.รัตติกร วรากูลศิริพันธุ์	
รศ.ดร.โยธิน เปรมปราณีรัชต์	

ค่าระดับคะแนนรวมที่เป็นเอกฉันท์จากคณะกรรมการสอบ O : Outstanding (ดีเยี่ยม)
วัน/เดือน/ปี ที่สอบ 29 สิงหาคม 2533 **เวลา** 9.00 น.
สถานที่สอบ ห้องประชุมคณะวิศวกรรมศาสตร์ (A-305) ตึก 6 ชั้น

บัณฑิตวิทยาลัยรับรองแล้ว

รองผู้อำนวยการฝ่ายวิชาการ
รักษาราชการแทนคณบดีบัณฑิตวิทยาลัย

วันที่ 30 เดือน สิงหาคม พ.ศ. 2533

สารบัญ

สารบัญภาพประกอบและตาราง

หน้า

หัวข้อวิทยานิพนธ์ การลดทอนตัวแบบในระบบเชิงเส้น
นักศึกษา นาย สัญญา สายะสถิตย์
อาจารย์ที่ปรึกษา รองศาสตราจารย์ วิพันธ์ ปรีชาพานิช
ระดับการศึกษา วิศวกรรมศาสตร์มหาบัณฑิต สาขาวิศวกรรมไฟฟ้า
ปีการศึกษา 2533

บทคัดย่อ

การลดทอนตัวแปรสถานะของสมการเสตทสเปซในโดเมนของเวลา และการลดทอน Order ของ Transferfunction ในโดเมนของความถี่ โดยการสร้างตัวแบบที่เรียกว่า Model Reduction การเลือกสถานะ การหาค่าพารามิเตอร์ และการหาค่าสัมประสิทธิ์ ของ Model Reduction มีหลายวิธีด้วยกัน แต่ละวิธีมีการคำนวณที่ยุ่งยากซับซ้อนมาก ทำให้การสร้าง Model Reduction กระทำได้ไม่สะดวก

วิทยานิพนธ์ฉบับนี้ได้เสนอการนำเอาไมโครคอมพิวเตอร์มาช่วยในการสร้าง Model Reduction โดยในโดเมนของเวลาได้เสนอถึงการพิจารณาพลังงานอิมพัลส์ของเอาท์พุตร่วมกับค่าไอเก็นของระบบในการเลือกสถานะ ส่วนการหาค่าพารามิเตอร์ได้เสนอการประยุกต์วิธี Singular Perturbation สำหรับในโดเมนของความถี่ การหาค่าสัมประสิทธิ์ของ Model Reduction ได้เสนอวิธี Fitting Time - Moments โดยนำการประมาณค่าแบบ Pade' มาช่วยในการแก้ปัญหาเกี่ยวกับการคำนวณค่า Time - Moments และทำการพัฒนาโปรแกรมเพื่อใช้กับไมโครคอมพิวเตอร์ตามวิธีการที่ได้เสนอ ซึ่งทำให้การสร้าง Model Reduction กระทำได้สะดวกขึ้น

THESIS MODEL REDUCTION IN LINEAR SYSTEMS

NAME SANYA SAYASATIT

THESIS ADVISOR Assoc. VIPAN PRIJAPANIJ

LEVEL OF STUDY MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

ACADEMIC YEAR 1990

ABSTRACT

The reduction of state variables of state equation in time domain and the reduction of order of transfer function in frequency domain, the method use is call "Model Reduction". There are several techniques in state's selection and evaluation of either parameters or cofficient. The calculation concerned are very complicated, resulting in model reduction which is not practical.

This thesis proposes a method to construct a model reduction with an aid of microcomputer. In time domain, impulse energy of output and eigen values of system are chosen for state considertion. An application of Singular Perturbation is proposed as a method of parameter evaluation. In the case of frequency domain, cofficients of the model reduction are obtain by a method call "Fitting Time - Moments". Pade's approximation method gives a lot of help in calculation of time - moments values. In the thesis software on microcomputer is developed, resulting in a much more easier construction of model reduction.

บทที่ 1
บทนำ

ปัจจุบันนี้ การวิเคราะห์ระบบควบคุมที่มีขนาดใหญ่ เมื่อทำการวิเคราะห์เป็นสมการเสตทสเปซแล้ว มักจะเกิดปัญหาเกี่ยวกับจำนวนของตัวแปรสถานะ (State Variables) ซึ่งจะมีจำนวนมากมายหลายสถานะ ในขณะเดียวกัน ถ้าหากทำการวิเคราะห์ค่า transfer function เพื่อจะศึกษาระบบในโดเมนของความถี่ ก็จะได้จำนวน Order ของสมการ transfer function ที่มีค่ามาก ทำให้การวิเคราะห์ระบบกระทำได้ยุ่งยากมาก จากปัญหาอันนี้ จึงเกิดแนวความคิดที่จะสร้างตัวแบบขึ้นใช้แทนระบบเดิม โดยที่ตัวแบบของระบบที่สร้างขึ้นใหม่นี้จะต้องมี Order ขนาดเล็กกว่าระบบเดิม ขณะเดียวกัน เอาท์พุตของระบบทั้งสองจะต้องใกล้เคียงกันมากที่สุด สำหรับตัวแบบของระบบที่สร้างใหม่นี้เรียกว่า "Reduced Model"

การวิเคราะห์ Reduced Model นี้ ได้มีผู้ที่สนใจศึกษาวิธีการต่างๆ ไว้เป็นจำนวนมาก ทำให้ได้ Reduced Model หลายรูปแบบด้วยกัน ขึ้นกับคณิตศาสตร์ที่ใช้ โดยที่วิธีการในโดเมนของเวลาส่วนใหญ่จะใช้วิธี Geometrical Techniques [3,6,33], Moment Matching Techinques [1,21] หรือวิธี Eigenvalue Preservation Techniques [2,10,11] ซึ่งวิธีสุดท้ายนี้เป็นวิธีที่ใช้หลักการของทฤษฎีเมตริกซ์เกี่ยวกับค่าไอเก็น และเป็นวิธีที่นิยมมากที่สุด เพราะเป็นวิธีที่เข้าใจง่าย มีนักวิจัยหลายท่านได้นำวิธีการนี้ไปพัฒนาหลายวิธีด้วยกัน [6,13,23] ส่วนวิธีการวิเคราะห์ Reduced Model ในโดเมนของความถี่นั้นส่วนใหญ่จะใช้ทฤษฎีการประมาณค่า (Approximation Theory) เช่น Routh Approximation [17] Pade´ Approximation [4,7,9,14,16, 34,35], Chebyshev Approximation [5] หรือวิธีการประมาณค่าอื่น ๆ [26,27,31] ซึ่งวิธีการประมาณค่านี้จะสามารถลด Order ของ Transfer function ได้

วิทยานิพนธ์ฉบับนี้ ได้แสดงการวิเคราะห์ Reduced Model ในระบบเชิงเส้นที่ไม่ขึ้นกับเวลา ทั้งในโดเมนของเวลา และในโดเมนของความถี่ โดยเสนอเฉพาะวิธีการที่

เข้าใจง่ายและเหมาะสมที่จะพัฒนาเป็นโปรแกรมสำเร็จรูปต่อไป

เนื้อหาของวิทยานิพนธ์ได้แบ่งออกเป็น 6 บท คือ

บทที่ 2 กล่าวถึงแนวความคิดทั่วไป ประวัติความเป็นมา เหตุผลที่จำเป็น เทคนิคและวิธีการต่างๆ ที่พยายามจะลดรูปแบบของระบบ ความหมายของ Reduced Model ตลอดจนปัญหาที่เกิดขึ้น

บทที่ 3 แสดงถึงทฤษฎี เทคนิคต่างๆ ในการลดทอนตัวแบบในโดเมนของเวลา การแก้ปัญหาเกี่ยวกับรูปแบบและสถานะที่จะคงไว้ใน Reduced Model การหาค่าพารามิเตอร์ของระบบในสมการเสตทสเปซของ Reduced Model โดยเสนอผลการศึกษาวิธี Aggregation และ Singular Perturbation วิธีการวิเคราะห์เงื่อนไขสถานะเริ่มต้น (initial condition) ของเวคเตอร์สถานะในสมการเสตทสเปซของ Reduced Model และแสดงถึงค่าความผิดพลาดของเอาท์พุตที่จะเกิดขึ้นหลังจากการลดทอนตัวแบบแล้ว

บทที่ 4 แสดงถึงทฤษฎีและเทคนิคที่นำมาประยุกต์ใช้ร่วมกับทฤษฎีการประมาณค่าแบบ Pade´ (Pade´ Approximation) ในการลดทอน Order ของสมการ transfer function ของระบบเพื่อสร้างเป็น Reduced Model ในโดเมนของความถี่ แสดงถึงวิธีการคำนวณค่าสัมประสิทธิ์และการแก้ปัญหาเกี่ยวกับความมีเสถียรภาพของ Reducde Model

บทที่ 5 เป็นการทดสอบระบบ โดยได้แสดงตัวอย่างที่นำมาทดสอบระบบทั้งในโดเมนของเวลาและในโดเมนของความถี่ โดยแสดงการหาค่าพารามิเตอร์การเปรียบเทียบผลตอบสนองของระบบเดิมและระบบที่แทนด้วย Reduced Model

บทที่ 6 สรุปและวิเคราะห์ผลตอบสนองของระบบที่แทนด้วย Reduced Model เมื่อเทียบกับระบบเดิม

บทที่ 2

ความเป็นมาและความหมายของ Reduced Model

ในบทนี้จะกล่าวถึงประวัติความเป็นมา เรื่องราวโดยทั่วไป และเหตุผลความจำเป็นในการที่จะลดทอนตัวแบบเพื่อสร้างเป็น Reduced Model แสดงถึงความหมายของ Reduced Model ตลอดจนถึงปัญหาในการสร้าง Reduced Model

2.1 ประวัติความเป็นมาและเหตุผลจำเป็น

การวิเคราะห์และการศึกษาระบบควบคุมสมัยใหม่ สำหรับระบบเชิงเส้นที่ไม่ขึ้นกับเวลา (Linear Time - Invariant Systems) อาจแทนด้วยสมการอนุพันธ์ คือ

$$(D^{(n)} + a_{n-1}D^{(n-1)} + \ldots + a_1D + a_o)y(t) = (b_{n-1}D^{(n-1)} + b_{n-2}D^{(n-2)} + \ldots + b_1D + b_o)u(t) \qquad (2.1)$$

เมื่อกำหนดให้ u(t) และ y(t) คืออินพุตและเอาท์พุตตามลำดับ และ $D^{(n)} = d^n/dt^n$, จาก สมการ (2.1) ถ้า $u(t) = (D^{(n)} + a_{n-1}D^{(n-1)} + \ldots + a_1D + a_o)z(t)$ จะได้ $y(t) = (b_{n-1}D^{(n-1)} + b_{n-2}D^{(n-2)} + \ldots + b_1D + b_o)z(t)$

ถ้าให้ $x_n(t)$ เป็นตัวแปรสถานะ มีนิยามคือ $x_1(t) = z(t)$, $x_2(t) = Dz(t)$, $x_3(t) = D^{(2)}z(t)$, ... ,$x_n(t) = D^{(n-1)}z(t)$ จะทำให้สมการ (2.1) เขียนในรูปแบบเมตริกซ์ได้คือ

$$d/dt\begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_{n-1}(t) \\ x_n(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \ldots & 0 \\ 0 & 0 & 1 & 0 \ldots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & 0 & \ldots\ 0 & 1 \\ -a_o & -a_1 & \ldots\ldots\ldots & & a_n \end{bmatrix}\begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_{n-1}(t) \\ x_n(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(t)$$

โดยที่มีสมการเอาท์พุตเป็น $y(t) = [\ b_o\ b_1\ \ldots\ b_n]\ x(t)$

เขียนให้อยู่ในรูปแบบอย่างง่ายได้เป็น

$$\dot{x}(t) = Ax(t) + Bu(t) \qquad (2.2)$$

$$y(t) = Cx(t) \qquad (2.3)$$

สมการที่ (2.2) เรียกว่า สมการสถานะ (State Equation) และ $x(t)$ เรียกว่าเวคเตอร์สถานะ (State Vector) โดยที่ A, B และ C เป็นเมตริกซ์ที่มีขนาด n*n, n*1 และ 1*n ตามลำดับ

สำหรับการวิเคราะห์ระบบในโดเมนของความถี่ สมการที่แสดงค่า transfer function หาได้จากการแปลงลาปลาซของสมการที่ (2.1) คือ

$$Y(s)/U(s) = G(s) = \frac{b_{n-1}s^{n-2} + \ldots + b_1 s + b_o}{s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_o} \qquad (2.4)$$

เมื่อ Y(s) และ U(s) เป็นการแปลงลาปลาซของ y(t) และ u(t) ตามลำดับ

ในกรณี ระบบที่วิเคราะห์เป็นระบบขนาดใหญ่ เมื่อทำการวิเคราะห์แล้ว จะได้จำนวนตัวแปรสถานะมากมายหลายร้อยสถานะ เมตริกซ์ที่ปรากฏในสมการสถานะจะมีมิติขนาดใหญ่มาก และถ้าวิเคราะห์ในโดเมนของความถี่ก็จะได้สมการแสดงค่า transfer function ที่มี Order สูง ๆ การวิเคราะห์ระบบเหล่านี้กระทำได้ยากมาก ดังนั้นจึงเกิดแนวความคิดที่จะต้องหารูปแบบที่จะลดรูปเพื่อความสะดวกในการวิเคราะห์ และศึกษาผลตอบสนองของระบบ โดยที่ผลตอบสนองของตัวแบบใหม่นี้จะต้องมีค่าต้องใกล้เคียงกับระบบเดิมมากที่สุด ตัวแบบใหม่ที่ได้นี้เราเรียกว่า "Reduced Model"

แนวความคิดเกี่ยวกับ Reduced Model นี้ ได้รับความสนใจอย่างแพร่หลาย เมื่อ DAVISON [10] ได้เสนอวิธีการลดขนาดของเมตริกซ์ในสมการสถานะเมื่อปี ค.ศ. 1966 โดยเทคนิคที่นำเสนอนี้เป็นวิธีการศึกษาระบบจากค่าไอเก็น (Eigen Values) ซึ่งต่อมาวิธีการนี้ เรียกว่า "เทคนิคของ DAVISON" [22] หลังจากนั้นได้มีนักวิจัยหลายท่านนำเอาเทคนิคอันนี้ไปพัฒนาและปรับปรุงขึ้นเป็นจำนวนมากมายหลายวิธี ทั้งในโดเมนของเวลา [2,6,13,15,22,23] และโดเมนของความถี่ [1,19,21,27] โดยการ

วิเคราะห์หา Reduced Model นี้ จะอยู่บนรากฐานทฤษฎีทางคณิตศาสตร์ และทฤษฎีระบบควบคุม ตัวแบบที่ได้จะเป็นรูปแบบทางคณิตศาสตร์ ตัวแบบของ Reduced Model ที่ได้นี้จะใช้แทนระบบเดิม ได้ดีเพียงใด ก็จะต้องศึกษาจากผลตอบสนองว่าใกล้เคียงกับระบบเดิมมากน้อยเพียงใด

2.2 ปัญหาในการวิเคราะห์ Reduced Model

ในหัวข้อที่แล้ว เราจะเห็นว่าเมื่อเราลดทอนตัวแบบเพื่อสร้าง Reduced Model ปัญหาสำคัญที่จะเกิดขึ้นในโดเมนของเวลาและในโดเมนของความถี่คือ

1. ตัวแปรสถานะที่จะคงไว้ใน Reduced Model ควรเป็นอะไร
2. รูปแบบพารามิเตอร์ของ Reduced Model ควรมีค่าเป็นอย่างไร
3. ค่าสัมประสิทธิ์ของตัวตั้งและตัวหารของสมการ transfer function ควรจะมีค่าเท่าใด

ปัญหาสองข้อแรกนั้นจะเกิดขึ้นเมื่อเราพยายามจะลดตัวแปรสถานะของสมการสถานะในการสร้าง Reduced Model ในโดเมนของเวลา สำหรับปัญหาข้อสุดท้ายนั้นเป็นปัญหาที่เกิดจากการลดค่า Order ของ transfer function ที่ได้จากการศึกษาระบบในโดเมนของความถี่

วิธีการแก้ปัญหาในข้อแรกนั้นส่วนใหญ่จะใช้การพิจารณาจากค่าไอเก็น หรือ สมการ Characteristic เป็นหลักสำคัญ ส่วนปัญหาในข้อที่สองนั้น จะใช้ทฤษฎีคณิตศาสตร์เมตริกซ์ชั้นสูงในการแก้ปัญหา สำหรับการแก้ปัญหาในข้อที่สาม จะใช้ทฤษฎีการประมาณค่าทางคณิตศาสตร์มาช่วยในการวิเคราะห์ ส่วนรายละเอียดในการแก้ปัญหาทั้งหมดนั้น จะแสดงไว้ในบทที่ 3 และบทที่ 4 ต่อไป

บทที่ 3
การวิเคราะห์การลดทอนตัวแบบในโดเมนของเวลา

วิธีการลดทอนตัวแบบเพื่อสร้าง Reduced Model ในโดเมนของเวลามีหลายวิธี เช่น วิธี Aggregation [2,3,15] วิธีของ Mellichamp [6] วิธีของ DAVISON [10,11,22] วิธี Signal flow [6,33] และอื่นๆ วิธีการต่างๆ นี้มีความยุ่งยาก ซับซ้อน ข้อดีข้อเสียต่างๆ กัน

สำหรับบทนี้ในวิทยานิพนธ์จะเสนอผลการศึกษาวิธี Singular Perturbation [6,13,18,23,32] เพราะเป็นวิธีที่ง่ายต่อการพัฒนาเป็นโปรแกรมสำเร็จรูป ส่วนการเลือกสถานะที่จะคงไว้ได้เสนอวิธีการพิจารณาพลังงานอิมพัลซ์ของเอาท์พุตร่วมกับค่าไอเก็นเป็นสำคัญ

3.1 ความหมายทางคณิตศาสตร์ของ Reduced Model

ในระบบเชิงเส้นที่ไม่ขึ้นกับเวลา สามารถแทนได้ด้วยสมการเสตทสเปซคือ

$$\dot{x} = Ax + Bu \qquad ; x \in R^n , u \in R^m \qquad (3.1)$$

$$y = Cx \qquad ; y \in R^p \qquad (3.2)$$

เมื่อทำการลดทอนตัวแบบเพื่อสร้าง Reduced Model ซึ่งมีขนาดเล็กกว่าตัวแบบของระบบเดิมแทน ด้วยสมการดังนี้ [10,22]

$$\dot{z} = Fz + Gu \qquad ; z \in R^k \qquad (3.3)$$

$$y^* = Hz \qquad ; y^* \in R^p \qquad (3.4)$$

โดยที่ k จะต้องมีค่าน้อยกว่า n และ y^* จะต้องมีค่าใกล้เคียงกับ y มากที่สุด

จากสมการที่ (3.1) - (3.4) จะให้ความหมายทางคณิตศาสตร์ของ Reduced Model ได้ว่า "เป็นการลดมิติของสมการเสตทสเปซ" นั่นเอง

3.2 <u>ทฤษฎี Aggregation</u>

ก่อนที่จะแสดงถึงรูปแบบของ Reduced Model โดยวิธี Singular Perturbation จำเป็นต้องเข้าใจถึงแนวความคิดในวิธี Aggregation เสียก่อน เพราะวิธีการทั้งสองนี้มีแนวความคิดที่ใกล้เคียงกันมาก

ความหมายทางคณิตศาสตร์ของการลดทอนตัวแบบในโดเมนของเวลาจากหัวข้อ 3.1 ทำให้ทราบว่า จุดมุ่งหมายหลักคือเราต้องการตัวแบบที่มีขนาดเล็กลงมากๆ ที่จะใช้แทนตัวแบบของระบบเดิม

โดยวิธีการ Aggregation จะใช้การแปลงเชิงเส้น ทำให้ได้เวคเตอร์สถานะ z ในสมการที่ (3.3) และ (3.4) มีค่าสัมพันธ์กับ เวคเตอร์สถานะ x ของระบบเดิม คือ

$$z = Kx \qquad ; z \in R^{k} \qquad (3.5)$$

เมตริกซ์ K นี้จะมีมิติเท่ากับ k*n เมื่อพิจารณาจากสมการที่ (3.4) และ (3.5) จะพบว่า y^{*} จะใกล้เคียงกับ y เพียงใด ก็ขึ้นกับเมตริกซ์ K และเมตริกซ์ K นี้เรียกว่า "เมตริกซ์ Aggregation"[2,18,22]

เมื่อเปรียบเทียบสมการที่ (3.1) และ (3.5) จะได้เป็น

$$K^{-1}z = AK^{-1}z + Bu$$

$$z = KAK^{-1}z + KBu \qquad (3.6)$$

สมการที่ (3.6) เมื่อนำไปเปรียบเทียบกับสมการที่ (3.3) จะได้ค่าพารามิเตอร์ของ Reduced Model โดยวิธี Aggregation เป็น

$$F = KAK^{-1} \tag{3.7}$$

$$G = KB \tag{3.8}$$

ส่วนพารามิเตอร์ H ในสมการที่ (3.4) อาจหาได้จากเงื่อนไขที่ว่า y^* จะต้องมีค่าใกล้เคียงกับ y มากที่สุด ทำให้เราสามารถประมาณค่าพารามิเตอร์ H ได้เป็น

$$HK \simeq C \tag{3.9}$$

จะเห็นว่าการหาค่าพารามิเตอร์โดยวิธีนี้ จำเป็นต้องหาค่าอินเวอร์สของเมตริกซ์ K ซึ่งจะหาได้ก็ต่อเมื่อ K เป็นเมตริกซ์จตุรัส (Square Matrix) นั่นคือ k จะต้องมีค่าเท่ากับ n แต่จากจุดมุ่งหมายของ Reduced Model ที่ต้องการจะลดขนาดของเมตริกซ์ที่ปรากฏในสมการเสตทสเปซ ทำให้มีเงื่อนไขว่า k จะต้องมีค่าน้อยกว่า n และจากเงื่อนไขนี้ทำให้เกิดปัญหาว่าไม่สามารถหาค่าอินเวอร์สของเมตริกซ์ K ได้ ทำให้ต้องนิยามเมตริกซ์ขึ้นใหม่คือ

ถ้า $KAK^{+} = KA$

แล้ว $K^{+} = K^{T}(KK^{T})^{-1}$ (3.10)

K^{+} นี้เราเรียกว่า "Moore - Penrose pseudoinverse" ของ K [22]

สมการที่ (3.10) อยู่ภายใต้การพิจารณาที่ว่าแรงค์ของ K = r และเมื่อนิยาม K^{+} จากสมการที่ (3.10) แล้ว พารามิเตอร์ F จากสมการที่ (3.7) จะได้เป็น [2,18,22]

$$F = KAK^{T}(KK^{T})^{-1} \tag{3.11}$$

เมื่อเราแทนค่า F, G และ H ที่ได้ลงในสมการที่ (3.3) และ (3.4) แล้ว เราก็จะได้รูปแบบของ Reduced Model ตามวิธี Aggregation ตามต้องการ

3.3 การคำนวณเมตริกซ์ Aggregation

จากหัวข้อที่แล้ว ถึงแม้ว่าเราจะได้รูปแบบของ Reduced Model แต่ก็จะพบว่าปัญหาที่ตามมาคือ เราจะสามารถคำนวณค่าเมตริกซ์ Aggregation ได้อย่างไร ในหัวข้อนี้จะแสดงถึงขั้นตอนในการคำนวณค่า เมตริกซ์ Aggregation [2,6,22] โดยอาศัยทฤษฎีการแปลงแบบ Similarity (Similarity tranformation)

กำหนดให้

$$x = Md \tag{3.12}$$

เมื่อ M คือ เมตริกซ์แบบ modal ของ A ในสมการสถานะ

สตมภ์ ของเมตริกซ์ M นี้มีการจัดเรียงจากซ้ายไปขวาสอดคล้องกับค่าไอเก็น [22] และเมื่อแทนค่า x ในสมการที่ (3.12) ลงในสมการที่ (3.1) จะได้

$$\dot{d} = \Lambda d + \Gamma u \tag{3.13}$$

โดยที่

$$\Lambda = M^{-1}AM$$

และ

$$\Gamma = M^{-1}B$$

จะเห็นว่าสมการที่ (3.13) ได้จากการแปลงแบบ Similarity นั่นเอง และถ้าค่าไอเก็นของเมตริกซ์ A มีค่าไม่ซ้ำกัน เมตริกซ์ Λ จะอยู่ในรูปของ Canonical

(Canonical form) แต่ถ้ามีค่าไอเก็นซ้ำกัน เมตริกซ์ Λ จะอยู่ในรูปแบบของจอร์แดน (Jordan Canonical form) เพื่อความสะดวกและง่ายต่อการทำความเข้าใจการพิสูจน์ในวิทยานิพนธ์นี้จะสมมุติให้ค่าไอเก็นของเมตริกซ์ A มีค่าไม่ซ้ำกัน

สมมุติให้ค่าไอเก็น l ค่าแรกเป็นค่าที่เราจะคงไว้ใน Reduced Model ตามวิธีของ DAVISON [10,11] เราจะได้การแปลงเชิงเส้นของเมตริกซ์ตัวใหม่เป็น

$$w = Td \tag{3.14}$$

โดยที่

$$T = [\, I_l \,:\, \emptyset \,]_{l \times n}$$

เมื่อ w คือเวคเตอร์สถานะของ Reduced Model ที่มีมิติเท่ากับ l เมื่อพิจารณาจากรูปแบบของ Reduced Model โดยวิธี Aggregation จากหัวข้อที่ผ่านมาจะได้ว่า

$$w = T \Lambda T^{T} w + T \Gamma \tag{3.15}$$

การเปลี่ยนการแสดงเมตริกซ์ model ไปเป็นรูปแบบทั่วไปของ Reduced Model โดยใช้การแปลงเช่นเดียวกับสมการที่ (3.12) จะได้

$$z = M_o w \tag{3.16}$$

โดยที่การแปลงเมตริกซ์ M_o หาได้โดย ให้ l สดมภ์แรกของเมตริกซ์ M มีรูปแบบเป็น

$$\begin{bmatrix} & & & \\ V_1^1 & V_1^2 & \cdots & V_1^1 \\ V_2^1 & V_2^2 & \cdots & V_2^1 \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ V_n^1 & V_n^2 & \cdots & V_n^1 \end{bmatrix}$$

เมตริกซ์ M_o จะเป็นเมตริกซ์ที่คงไว้ใน Reduced Model โดยเลือกสถานะจากเมตริกซ์ M ตัวอย่างเช่น ถ้าเราเลือกตัวแปรสถานะที่จะคงไว้เป็น x_1, x_4, x_{n-1} จะได้ M_o มีรูปแบบเป็น

$$M_o = \begin{bmatrix} V_1^1 & V_1^2 & V_1^3 \\ V_2^1 & V_2^2 & V_2^3 \\ V_{n-1}^1 & V_{n-1}^2 & V_{n-1}^3 \end{bmatrix}$$

เมื่อได้ค่า M_o และจากการแปลงในสมการที่ (3.16) จะได้สมการสถานะของ Reduced Model เป็น [23]

$$z = M_o T \Lambda T^T M^{-1} z + M_o T M^{-1} B u \qquad (3.17)$$

เมื่อแทนค่า d จากสมการที่ (3.12) ลงในสมการที่ (3.14) จะได้

$$w = TM^{-1}x \qquad (3.18)$$

แทนค่า w จากสมการที่ (3.18) ลงในสมการที่ (3.16) จะได้เวคเตอร์สถานะของ Reduced Model (z) เป็น

$$z = M_o w = M_o T M^{-1} x \qquad (3.19)$$

เปรียบเทียบสมการที่ (3.19) กับสมการที่ (3.5) จะได้ค่าเมตริกซ์ Aggregation เป็น

$$K = M_o T M^{-1} \qquad (3.20)$$

สมการที่ (3.20) จะใช้ในการหาค่าเมตริกซ์ Aggregation จากนั้นจึงสามารถคำนวณค่าพารามิเตอร์ของ Reduced Model ตามวิธีการที่ผ่านมาได้ อย่างไรก็ตามมีผู้เสนอวิธีการต่างๆ ในการคำนวณค่าเมตริกซ์ Aggregation อีกหลายวิธีด้วยกัน [3, 18,32,33] นักวิจัยแต่ละท่านก็พยายามแสดงให้เห็นว่าวิธีการของตนเป็นวิธีการที่เหมาะสม ซึ่งวิธีการต่างๆ เหล่านี้ก็อาศัยแนวความคิดของ DAVISON และวิธีการที่ได้แสดงไว้แล้วเป็นส่วนใหญ่

3.4 วิธีการ Singular Perturbation

วิธีการ Singular Perturbation เป็นวิธีการหารูปแบบของ Reduced Model โดยอาศัยทฤษฎีทางเมตริกซ์ชั้นสูง ในการแบ่งเมตริกซ์ออกเป็น เมตริกซ์ย่อยๆ (Partitioning Matrix)

จากระบบเชิงเส้นที่ไม่ขึ้นกับเวลาเราสามารถแสดงสมการเสตทสเปซและสมการเอาท์พุตเป็นไปตามสมการที่ (3.1) และ (3.2) คือ

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

$$y = Cx$$

และถ้าเราแบ่งเมตริกซ์ในสมการที่ 3.1 และ 3.2 เป็น

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u \qquad (3.21)$$

$$y = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad (3.22)$$

จากสมการที่ (3.21) ถ้าเราเลือก x_1 เป็นเวคเตอร์สถานะที่จะคงไว้ใน Reduced Model โดยที่เราจะตัดเวคเตอร์สถานะ x_2 ทิ้งไป สมการที่ (3.21) จะสามารถแสดงเป็นรูปแบบใหม่ [6,32] ได้คือ

$$\begin{bmatrix} \dot{x}_1 \\ \epsilon\dot{x}_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u \qquad (3.23)$$

โดยที่ ϵ เป็นจำนวนบวกที่มีค่าน้อยมาก จากการประมาณค่าของระบบนี้ค่าของ ϵ จะมีค่าเข้าใกล้ 0 ทำให้สมการที่ (3.23) ได้เป็น

$$\begin{bmatrix} \dot{x}_1 \\ 0 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_{2ss} \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u \qquad (3.24)$$

โดยมีสมการเอาท์พุตเป็น

$$y = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_{2ss} \end{bmatrix} \qquad (3.25)$$

เมื่อ $x_{2ss} = \lim_{t \to \infty} x_2(t)$ x_{2ss} แสดงค่า steady - state ของ $x_2(t)$ ถ้าเราสมมุติให้ x_{2ss} สามารถหาค่าได้ (exists) และจากสมการ (3.24) เราสามารถหาค่า x_{2ss} ได้โดย

$$A_{22}x_{2ss} = - A_{21}x_1 - B_2u$$

$$x_{2ss} = - A_{22}^{-1}(A_{21}x_1 + B_2u) \qquad (3.26)$$

จากสมการที่ (3.26) นั้นมีเงื่อนไขที่สำคัญคือ เมตริกซ์ A_{22} จะต้องมีค่าอินเวอร์ส แต่อย่างไรก็ตามถ้า A_{22} ไม่มีอินเวอร์สเราจะต้องใช้ค่า pseudo - inverse ของ A_{22} แทน [32] ซึ่งแนวความคิดนี้ได้แสดงไว้แล้วในหัวข้อที่ 3.2

จากสมการที่ (3.24) เราจะได้เวคเตอร์สถานะเป็น

$$\dot{x}_1 = A_{11}x_1 + A_{12}x_{2ss} + B_1u \qquad (3.27a)$$

แทนค่า x_{2ss} ในสมการที่ (3.26) ลงในสมการที่ (3.27a) จะได้

$$\dot{x}_1 = A_{11}x_1 + A_{12}\left[-A_{22}^{-1}(A_{21}x_1 + B_2u) \right] + B_1u$$

$$\dot{x}_1 = \left[A_{11} - A_{12}A_{22}^{-1}A_{21} \right]x_1 + \left[B_1 - A_{12}A_{22}^{-1}B_2 \right] u \qquad (3.27b)$$

และจากสมการที่ (3.25)

$$y = C_1 x_1 + C_2 x_{2ss} \qquad (3.28)$$

แทนค่า x_{2ss} จากสมการที่ (3.26) ลงในสมการที่ (3.28)

$$y = C_1 x_1 + C_2 \left[-A_{22}^{-1}(A_{21} x_1 + B_2 u) \right]$$

$$y = \left[C_1 - C_2 A_{22}^{-1} A_{21} \right] x_1 - \left[C_2 A_{22}^{-1} B_2 \right] u \qquad (3.29)$$

จากสมการที่ (3.27) และสมการที่ (3.29) อาจเขียนให้เป็นรูปแบบทั่วไปของ Reduced Model ได้เป็น

$$\dot{x}_1 = Fx_1 + Gu \qquad (3.30)$$

$$y = Hx_1 + Lu \qquad (3.31)$$

สมการที่ (3.30) และ (3.31) คือรูปแบบของ Reduced Model โดยวิธี Singular Parturbation โดยที่

$$F = A_{11} - A_{12} A_{22}^{-1} A_{21} \qquad (3.32)$$

$$G = B_1 - A_{12} A_{22}^{-1} B_2 \qquad (3.33)$$

$$H = C_1 - C_2 A_{22}^{-1} A_{21} \qquad (3.34)$$

$$L = -C_2 A_{22}^{-1} B_2 \qquad (3.35)$$

วิธีการหารูปแบบและการประมาณค่าพารามิเตอร์ของ Reduced Model โดยวิธี

นี้เรียกว่า "Singular Perturbation"

จากแนวความคิดนี้จะเห็นได้ว่าเป็นวิธีที่ไม่สมบูรณ์ดีนัก เพราะยังไม่ได้มีการพิจารณาว่า จะใช้อะไรเป็นหลักในการพิจารณาว่าสถานะใดมีความสำคัญมากน้อยกว่ากันเพื่อจะได้เลือกสถานะที่สำคัญเอาไว้ และเมื่อสามารถเลือกตัวแปรสถานะได้แล้ว ค่าพารามิเตอร์ F, G, H และ L ของ Reduced Model อาจมีรูปแบบแตกต่างจากสมการที่ (3.32) ถึง (3.35) บ้างเล็กน้อย

3.5 การเลือกสถานะที่จะคงไว้ใน Reduced Model

จากการวิเคราะห์ Reduced Model โดยวิธี Aggregation เราจะพบว่ารูปแบบและค่าพารามิเตอร์ของ Reduced Model ขึ้นอยู่กับเมตริกซ์ Aggregation ส่วนในหัวข้อที่แล้วเราจะเห็นถึงรูปแบบอย่างง่ายๆ ของ โดยวิธี Singular Perturbation แต่ปัญหาสำคัญในการพิจารณาว่าควรจะเลือกตัวแปรสถานะใดที่จะคงไว้ใน Reduced Model ยังมิได้กล่าวถึง

ในการวิเคราะห์ระบบเพื่อสร้างเป็น Reduced Model นั้น การเลือกตัวแปรสถานะที่จะคงไว้ในตัวแบบที่จะสร้างขึ้นใหม่ เป็นปัญหาที่สำคัญมาก ได้มีผู้เสนอวิธีการไว้ไม่มากนัก [2,23] ทำให้ไม่สามารถระบุได้แน่นอนว่าวิธีการใดจะเป็นวิธีการที่ดีที่สุด

สำหรับในวิทยานิพนธ์นี้จะเสนอวิธีการพิจารณาพลังงานอิมพัลซ์ของเอาท์พุต และใช้คุณสมบัติเกี่ยวกับค่าไอเก็นมาช่วยในการวิเคราะห์

3.5.1 แนวความคิดในการพิจารณาค่าไอเก็น

ระบบเชิงเส้นที่ไม่ขึ้นกับเวลาเมื่อแสดงเป็นสมการเสตทสเปซแล้ว จะได้ตามสมการที่ (3.1) และ (3.2) สำหรับสมการที่ (3.1) นั้น เพื่อความสะดวกเราจะกำหนดให้เมตริกซ์ A เป็นเมตริกซ์เสถียร (Stable Matrix) และค่าไอเก็นของเมตริกซ์ A มีค่าไม่ซ้ำกัน ถ้าเราแปลงเมตริกซ์ A เป็นเมตริกซ์เส้นทะแยงมุม (Diagonal Matrix) จะได้

$$A = V \Lambda U \qquad (3.36)$$

เมื่อ V คือ Right Eigenvector

และ U คือ Left Eigenvector

โดยที่ $VU = I_n$ (3.37)

I_n คือเมตริกซ์ Identity ที่มีขนาด n*n

และเมตริกซ์ Λ คือ

$$\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \quad (3.38)$$

โดยที่เมตริกซ์ Λ นี้ มีการเรียงลำดับค่าไอเก็นคือ

$$|Re(\lambda_1)| < |Re(\lambda_2)| < \ldots < |Re(\lambda_n)| \quad (3.39)$$

กำหนดให้ ϕ คือ Transition Matrix มีนิยามว่า

$$\phi(t) = e^{At} \quad (3.40)$$

จากสมการที่ (3.1) และค่า Transition Matrix ในสมการที่ (3.40) ทำให้สามารถแก้สมการสถานะได้คือ

$$x(t) = \phi(t-t_o)x(t_o) + \int_{t_o}^{t} \phi(t-\tau)Bu(\tau)\, d\tau \quad (3.41)$$

และจากวิธี Similary Transform เมตริกซ์ A มีรูปแบบตามสมการที่ (3.36) ทำให้ Transition Matrix ในสมการที่ (3.40) มีรูปแบบใหม่คือ [23,32]

$$\phi(t) = Ve^{\Lambda t}U \quad (3.42)$$

เมื่อแทนค่า Transition Matrix จากสมการที่ (3.42) ลงในสมการที่ (3.41) และกำหนดให้ $t_o = 0$, $x(t_o) = 0$ จะได้เวคเตอร์สถานะเป็น [23]

$$x(t) = \int_o^t Ve^{\Lambda(t-\tau)}UBu(\tau)\,d\tau \qquad (3.43)$$

ถ้าให้อินพุตเป็นค่าคงที่ สมการที่ (3.43) จะได้คำตอบคือ [10,23]

$$x(t) = \begin{bmatrix} \dfrac{-VUBu}{\lambda_1} + \dfrac{Ve^{\lambda_1 t}UBu}{\lambda_1} \\ \dfrac{-VUBu}{\lambda_2} + \dfrac{Ve^{\lambda_2 t}UBu}{\lambda_2} \\ \vdots \\ \vdots \\ \dfrac{-VUBu}{\lambda_n} + \dfrac{Ve^{\lambda_n t}UBu}{\lambda_n} \end{bmatrix} \qquad (3.44)$$

เมื่อพิจารณาคำตอบในสมการที่ (3.44) เราจะพบว่าเมื่อค่าไอเก็นของสถานะใดมีค่ามาก จะทำให้ตัวแปรสถานะ ($x(t)$) มีค่าน้อย ดังนั้นเมื่อนำค่า $x(t)$ ไปหาค่าเอาท์พุตตามสมการที่ (3.2) จะเห็นว่าเอาท์พุตลดลง ดังนั้นเราจะสามารถตัดสถานะนั้นทิ้งไปได้ อย่างไรก็ดีเราจะพบว่าวิธีนี้เพียงแต่นำค่าไอเก็นของเมตริกซ์ A มาพิจารณาเท่านั้นไม่ได้พิจารณาถึงความสำคัญของเมตริกซ์ B และเมตริกซ์ C ทำให้วิธีนี้เกิดข้อผิดพลาดค่อนข้างมาก

3.5.2 Energy Integral Participation Matrices

จากแนวความคิดในการพิจารณาค่าไอเก็นของเมตริกซ์ระบบ จะเห็นได้ว่าไม่มีการนำเมตริกซ์ B และ C มาพิจารณา ในหัวข้อนี้จึงเสนอวิธีการพิจารณาพลังงานเอาท์พุต

ของระบบ โดยการนิยามเมตริกซ์ของพลังงาน โดยวิธีการนี้ เมตริกซ์ทุกเมตริกซ์ที่ปรากฏในสมการเสตทสเปซ จะถูกนำมาพิจารณาด้วย ทำให้ได้ผลตอบสนองของระบบ Reduced Model มีความถูกต้องมากขึ้น

พิจารณาจากสมการที่ (3.1) ถ้าให้อินพุต u(t) คือ unit impulse และ Similary Transform เมตริกซ์ A เป็นเมตริกซ์ทะแยงมุมจะได้คำตอบ impulse - response เป็น

$$x(t) = Ve^{\Lambda t}UB \qquad ; t > 0$$

ถ้าสมการเอาท์พุตมีรูปแบบเป็นไปตามสมการที่ (3.2) แล้ว พลังงานอิมพัลซ์ ของเอาท์พุตมีรูปแบบทั่วไปเป็น

$$E = \int_0^\infty y^T y \, dt \qquad (3.45)$$

เมื่อ superscript T คือ ทรานสโพสเมตริกซ์

แทนค่า y จากสมการที่ (3.1) ลงในสมการที่ (3.45) จะได้

$$E = \int_0^\infty x^T C^T C x \, dt \qquad (3.46)$$

ถ้ากำหนดให้

$$D = C^T C = [d_{ij}] \qquad (3.47)$$

แทนค่าเมตริกซ์ D จากสมการที่ (3.47) ลงในสมการที่ (3.46) ทำให้สมการพลังงานเป็น [23]

$$E_k = \sum_{j=1}^{n}\sum_{i=1}^{n} d_{ij} \int_0^\infty x_i x_j \, dt \qquad (3.48)$$

กำหนดให้เมตริกซ์ $W_k = [W^{(k)}]$ มีนิยามคือ

$$W_{ij} = \int_o^\infty x_i x_j \, dt \qquad (3.49)$$

เมตริกซ์ W_k นี้เรียกว่า "State Energy Integral Participation Matrix" [23]

จากค่า W ในสมการที่ (3.49) นี้ เมื่อนำไปแทนค่าในสมการที่ (3.48) จะได้

$$E_k = \sum_{j=1}^{n} \sum_{i=1}^{n} d_{ij} w_{ij} \qquad (3.50)$$

ถ้าเรานำผลคูณของเมตริกซ์ DW ไปเปรียบเทียบกับสมการที่ (3.50) เราจะพบว่าพลังงานเอาท์พุตมีค่าเท่ากับค่า trace ของเมตริกซ์ DW นั่นคือ

$$E_k = tr(DW_k) = tr(P) \qquad (3.51)$$

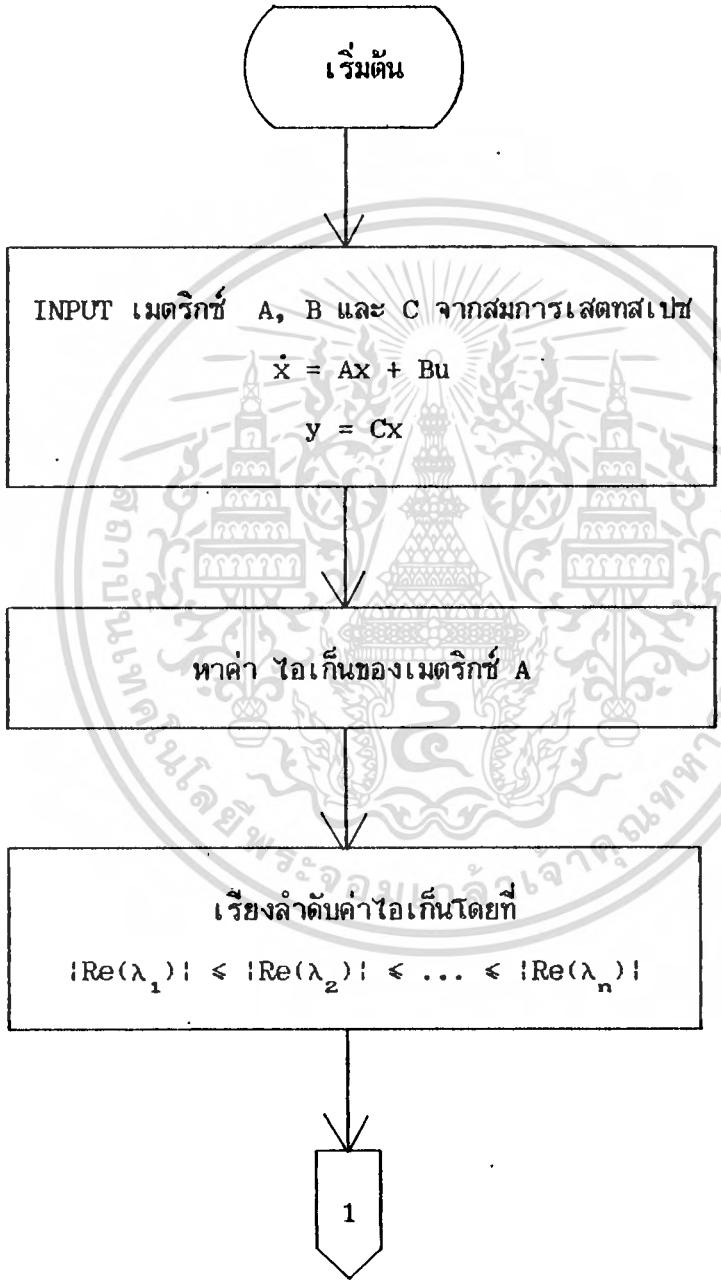
เมื่อ tr แทนค่า trace ของเมตริกซ์

เมตริกซ์ DW_k นี้เรียกว่า "Output Energy Integral Participation Matrix" [23]

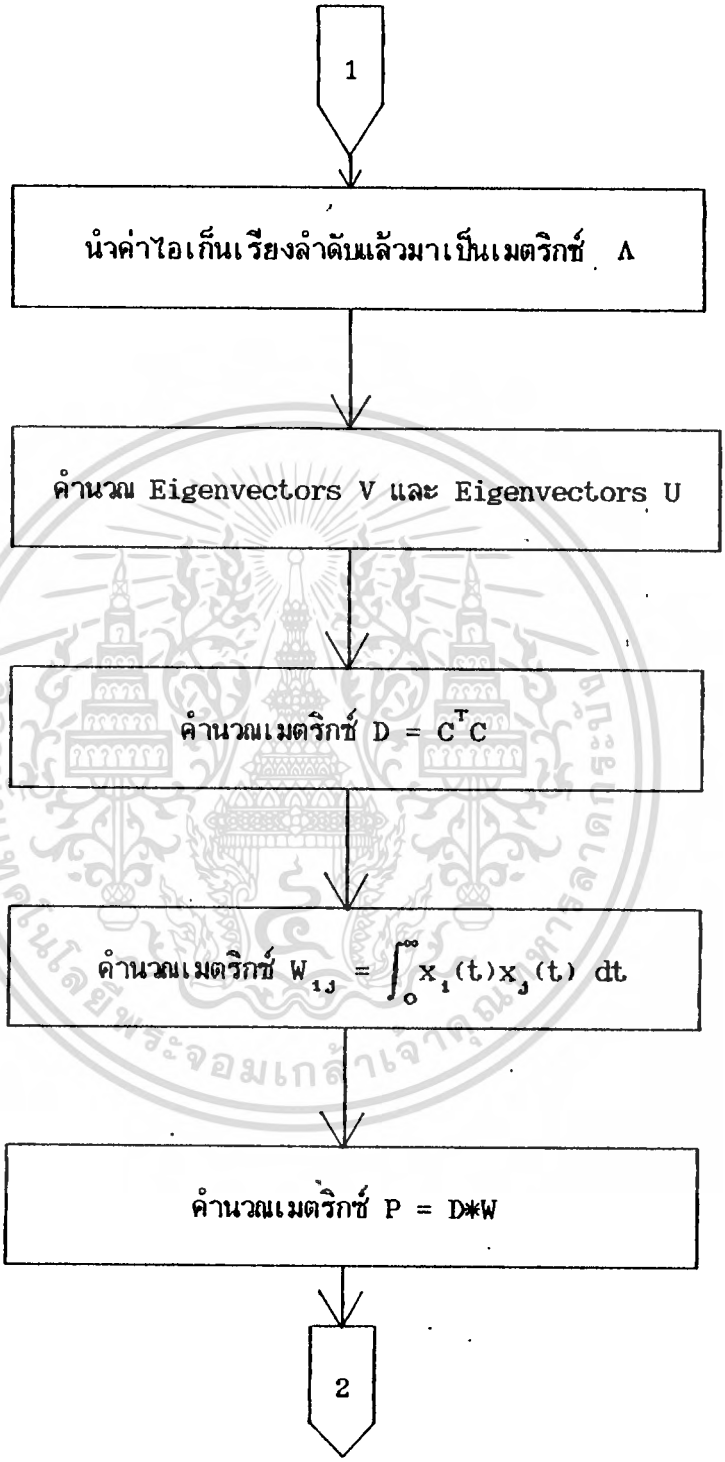
จากสมการที่ (3.51) นี้จะเห็นได้ว่า สถานะที่มีความสำคัญมากที่สุดของระบบเป็นสถานะที่เราจะคงไว้ใน Reduced Model จะสมนัยกับสมาชิกที่มีค่ามากที่สุดของเส้นทะแยงมุมหลัก (Principle Diagonal) ของเมตริกซ์ DW การเลือกสถานะโดยวิธีนี้ก็คือการเลือกสถานะที่มีพลังงานสูงสุดนั่นเอง

เมื่อพิจารณาวิธีการวิเคราะห์ในลักษณะนี้ เราจะพบว่ามีการนำเอาเมตริกซ์ B และ เมตริกซ์ C มาทำการวิเคราะห์ด้วย ทำให้การเลือกสถานะครอบคลุมพฤติการณ์ทั้งหมดของระบบ.

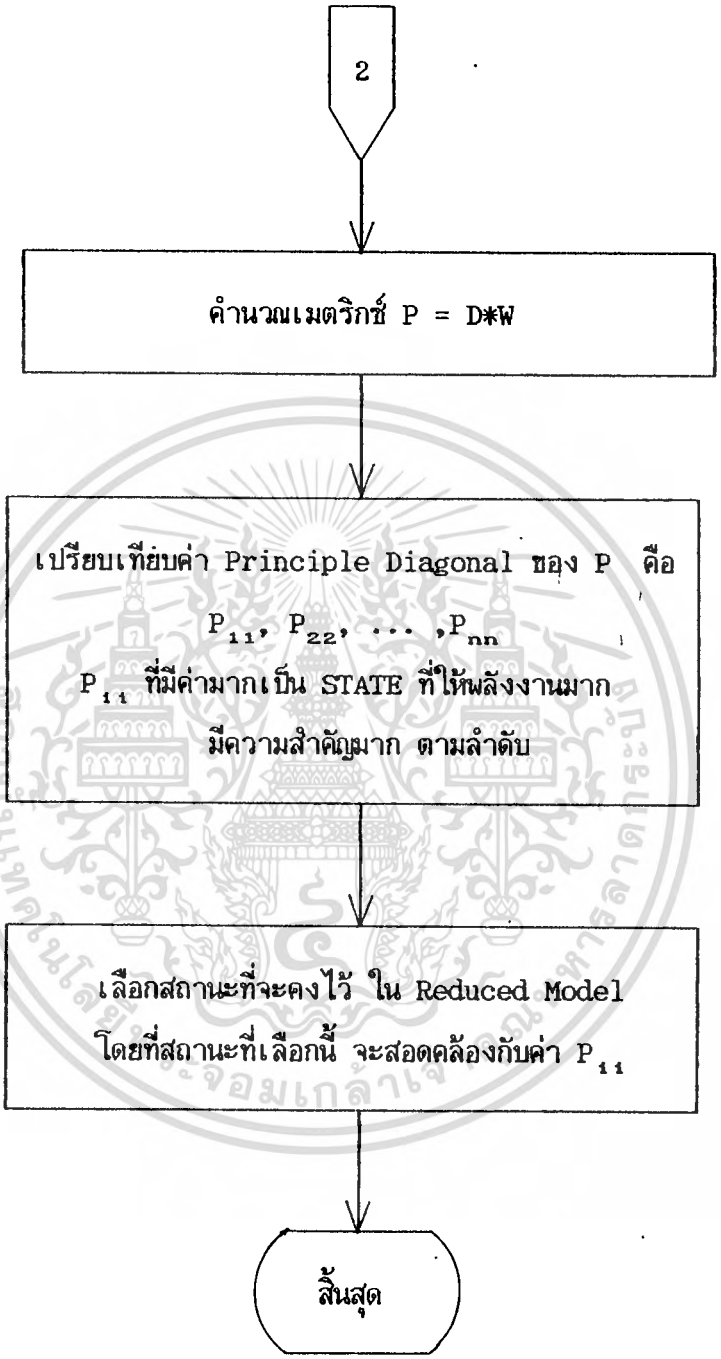
จากการวิเคราะห์ที่ผ่านมาเราจะสามารถสร้างเป็นขั้นตอนการคำนวณ (Algorithm) ได้ตามรูปที่ 1



รูปที่ 1 แสดงขั้นตอนการเลือกสถานะที่จะคงไว้ใน Reduced Model



รูปที่ 1 (ต่อ) แสดงขั้นตอนการเลือกสถานะที่จะคงไว้ใน Reduced Model



รูปที่ 1 (ต่อ) แสดงขั้นตอนการเลือกสถานะที่จะคงไว้ใน Reduced Model

3.6 <u>การวิเคราะห์พารามิเตอร์ของ Reduced Model</u>

ดังที่ได้แสดงไว้ในบทที่ 2 ว่า ปัญหาของการวิเคราะห์ Reduced Model สำหรับระบบเชิงเส้นในโดเมนของเวลา มีปัญหาที่สำคัญสองประการคือ การหารูปแบบและการเลือกสถานะที่จะคงไว้ใน Reduced Model จากหัวข้อที่แล้ว ได้เสนอวิธีการเลือกสถานะโดยการพิจารณาพลังงานอิมพัลซ์ของเอาท์พุตประกอบกับการพิจารณาค่าไอเก็นและเมื่อเลือกสถานะที่จะคงไว้ได้แล้ว ขั้นตอนต่อไปคือการแก้ปัญหาเกี่ยวกับรูปแบบของ Reduced Model

จากวิธี "Singular Perturbation" ในหัวข้อที่แล้วเราทราบว่าถ้าสมการเสตทสเปซแทนระบบเชิงเส้นที่ไม่ขึ้นกับเวลา มีรูปแบบเป็นไปตามสมการที่ (3.1) และ (3.2) คือ

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

$$y = Cx$$

จะได้สมการเสตทสเปซที่ใช้แทนระบบใน Reduced Model มีรูปแบบเป็นไปตามสมการที่ (3.30) และ (3.31) คือ

$$\dot{x} = Fx + Gu$$

$$y = Hx + Lu$$

โดยมีข้อกำหนดที่สำคัญสองประการคือ ขนาดของมิติใน Reduced Model ต้องมีขนาดเล็กกว่าระบบเดิมและเอาท์พุตของ Reduced Model จะต้องมีค่าใกล้เคียงกับระบบเดิมให้มากที่สุด

จากจุดนี้เราจะพบว่า การแก้ปัญหาของรูปแบบที่ควรจะเป็นหลังจากการเลือกสถานะของการลดทอนตัวแบบแล้วคือ การคำนวณค่าพารามิเตอร์ F, G, H และ L ที่

ปรากฏในสมการเสตทสเปซที่ใช้แทนระบบ Reduced Model นั่นเอง

สำหรับในหัวข้อนี้จะเสนอวิธีการคำนวณค่าพารามิเตอร์ F, G, H และ L โดยใช้ทฤษฎีการแบ่งเมตริกซ์ และวิธีการต่างๆ ที่แสดงไว้แล้วมาช่วยในการวิเคราะห์

จากสมการที่ (3.1) ถ้าเราแบ่งเมตริกซ์ x เป็น

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.52}$$

เมื่อ $x_1 \in R^r$

และ $x_2 \in R^{n-r}$

และถ้าเราแบ่งเมตริกซ์ต่างๆ ที่ปรากฏในวิธี Singular Perturbation เราจะได้สมการแทนระบบใหม่เป็น

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u \tag{3.53}$$

เราแบ่งเมตริกซ์ Λ และ U โดยวิธีเดียวกันจะได้

$$\Lambda = \begin{bmatrix} \Lambda_1 & \emptyset \\ \emptyset & \Lambda_2 \end{bmatrix} \tag{3.54}$$

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \qquad (3.55)$$

เมื่อ Λ มีข้อจำกัดเช่นเดียวกับสมการที่ (3.38) และ (3.39) และ

$$\Lambda_1 = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_r)$$

$$\Lambda_2 = \text{diag}(\lambda_{r+1}, \lambda_{r+2}, \ldots, \lambda_n)$$

คูณสมการที่ (3.53) ด้วย U

$$\begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$+ \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u \qquad (3.56)$$

ถ้าเรากำหนดตัวแปรใหม่คือเมตริกซ์ z โดยมีนิยามคือ

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad (3.57)$$

จากสมการที่ (3.57) และจากคุณสมบัติที่ว่าค่า Left Eigenvectors และ Right Eigenvectors เป็นอินเวอร์สซึ่งกันและกัน ทำให้หาค่าเมตริกซ์ x ในสมการที่ (3.57) ได้เป็น

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad (3.58)$$

นำค่าเมตริกซ์ z และเมตริกซ์ x จากสมการที่ (3.57) และ (3.58) แทนลงไปในสมการที่ (3.56) เราจะได้

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} \tilde{B}_1 \\ \tilde{B}_2 \end{bmatrix} u \qquad (3.59)$$

โดยที่

$$\begin{bmatrix} \tilde{B}_1 \\ \tilde{B}_2 \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \qquad (3.60)$$

จากจุดนี้เราสนใจคำตอบที่เป็น steady-state ของสมการที่ (3.59) เพราะตัวแปร z จะแทนตัวแปร x อยู่

ถ้าเราให้ u เป็นค่าของอินพุต u ที่ $t \to \infty$ นั่นคือ $\lim_{t \to \infty} u(t) = \bar{u}$

และถ้านิยามตัวแปร $\bar{z}_2$ เป็นค่า steady-state ของ z_2 จากการประยุกต์วิธี Singular Perturbation ร่วมกับการพิจารณาค่าไอเก็นและพลังงานเอาท์พุต เราสามารถกำหนดให้ $\dot{z}_2 = 0$ นำค่านี้แทนลงไปในสมการที่ (3.59) โดยให้ $u = \bar{u}$ เราจะสามารถหาคำตอบใน steady-state ของ z_2 ได้จาก

$$0 = \Lambda_2 \bar{z} + \tilde{B}_2 \bar{u}$$

$$\bar{z}_2 = - \Lambda_2^{-1} \tilde{B}_2 \bar{u} \qquad (3.61)$$

เพราะว่า $u(t) \rightarrow \bar{u}$ เราสามารถประมาณค่า z_2 ได้เป็น

$$z_f = - \Lambda_2^{-1} \tilde{B}_2 u \qquad (3.62)$$

เราจะใช้ค่า z_f ที่ได้ ทำการขจัดค่า z_2 ฉะนั้นในสมการที่ (3.57) สามารถเขียนใหม่ได้เป็น

$$\begin{bmatrix} z_1 \\ z_f \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} x_\rho \\ \tilde{x}_2 \end{bmatrix} \qquad (3.63)$$

เมื่อ x_ρ และ $\tilde{x}_2$ เป็นค่าประมาณของ x_1 และ x_2 ตามลำดับ

สมการที่ (3.63) เราสามารถขจัด z_f ได้โดยการแทนค่า z_f จากสมการที่ (3.62) ได้เป็น

$$U_{21} x_\rho + U_{22} \tilde{x}_2 = - \Lambda_2^{-1} \tilde{B}_2 u \qquad (3.64)$$

สำหรับการแก้สมการหา $\tilde{x}_2$ จากสมการที่ (3.64) นี้ มีเงื่อนไขที่สำคัญคือ เมตริกซ์ U_{22} ต้องเป็นเมตริกซ์ nonsingular เงื่อนไขนี้เป็นสิ่งที่จำเป็นมาก เพราะถ้าเมตริกซ์ U_{22} เป็นเมตริกซ์ singular แล้ว การเลือกสถานะที่จะไว้ใน Reduced Model จะต้องได้รับการพิจารณาใหม่

จากสมการที่ (3.64) แก้สมการหาค่า $\tilde{x}_2$ จะได้

$$\tilde{x}_2 = -U_{22}^{-1}\left[U_{21}x_\rho + \Lambda_2^{-1}\tilde{B}_2 u\right] \qquad (3.65)$$

และถ้า x_ρ และ $\tilde{x}_2$ เป็นคำตอบที่ได้จากการแก้สมการดิฟเฟอร์เรนเชียลในสมการที่ (3.53) เราจะสามารถแทนค่า x_ρ และ $\tilde{x}_2$ แทน x_1 และ x_2 ได้ จากนั้นเราแทนค่า $\tilde{x}_2$ ในสมการที่ (3.65) ลงไป เราจะได้ค่าของสมการสถานะของรูปแบบที่เราลดทอนตัวแบบเป็น

$$\dot{x}_\rho = \left[A_{11} - A_{12}U_{22}^{-1}U_{21}\right]x_\rho + \left[\tilde{B}_1 - A_{12}U_{22}^{-1}\Lambda_2^{-1}\tilde{B}_2\right]u \qquad (3.66)$$

เมื่อเปรียบเทียบกับสมการที่ (3.30) เราจะได้ค่าพารามิเตอร์ F และ G ของ Reduced Model เป็น

$$F = A_{11} - A_{12}U_{22}^{-1}U_{21} \qquad (3.67)$$

$$G = \tilde{B}_1 - A_{12}U_{22}^{-1}\Lambda_2^{-1}\tilde{B}_2 \qquad (3.68)$$

สำหรับสมการเอาท์พุต เราแบ่งเมตริกซ์ C เป็น

$$C = \begin{bmatrix} C_1 & C_2 \end{bmatrix}$$

โดยที่ C_1 มีมิติเท่ากับ m*r และ C_2 มีมิติเป็น m*(n-r) สมการเอาท์พุตจะมีรูปแบบใหม่เป็น

$$y_\rho = C_1 x_\rho + C_2 \tilde{x}_2 \qquad (3.69)$$

เมื่อแทนค่า $\tilde{x}_2$ จากสมการที่ (3.65) จะได้สมการเอาท์พุตหลังจากการลดทอนตัวแบบเป็น

$$y_\rho = \left[C_1 - C_2 U_{22}^{-1} U_{21} \right] x_\rho - C_2 U_{22}^{-1} \Lambda_2^{-1} \tilde{B}_2 u \qquad (3.70)$$

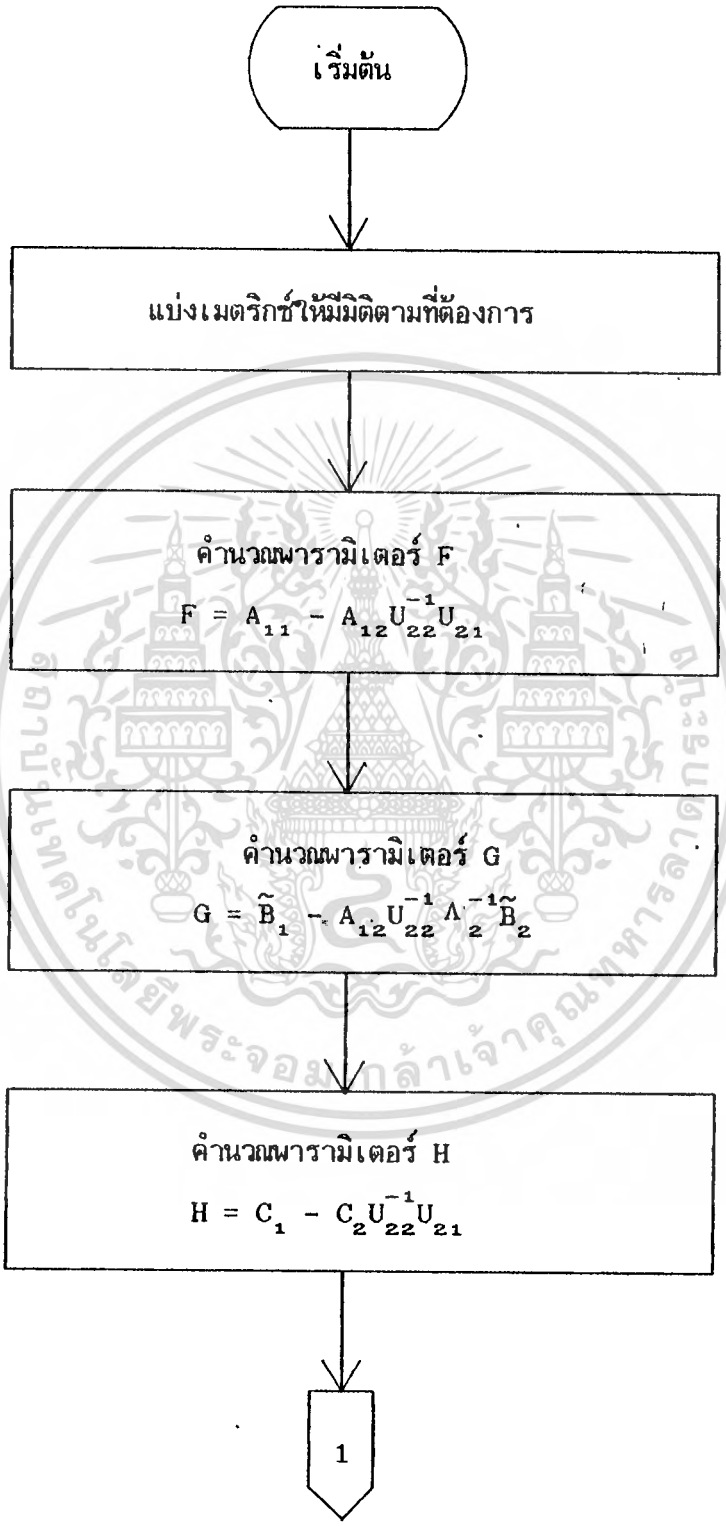
นำสมการที่ (3.70) ไปเปรียบเทียบกับสมการที่ (3.31) จะได้ค่าพารามิเตอร์ H และ L ของ Reduced Model เป็น

$$H = C_1 - C_2 U_{22}^{-1} U_{21} \qquad (3.71)$$

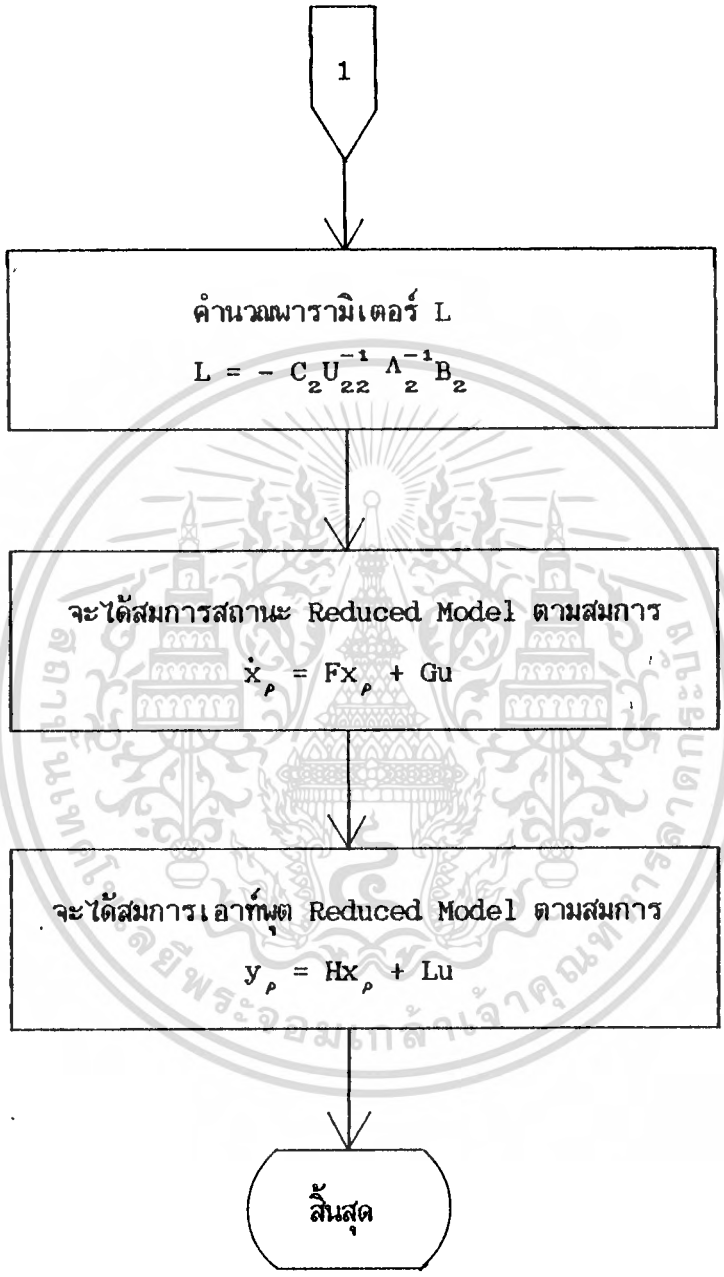
$$L = - C_2 U_{22}^{-1} \Lambda_2^{-1} \tilde{B}_2 \qquad (3.72)$$

เมื่อได้ค่าพารามิเตอร์ F, G, H และ L แล้ว เราก็จะได้ สมการเสตทสเปซที่ใช้แทนระบบ Reduced Model เป็นไปตามสมการที่ (3.30) และ (3.31) ที่ได้จากวิธีตาม Singular Perturbation ต้องการ

จากหัวข้อที่แล้ว ได้แสดงขั้นตอนในการเลือกสถานะที่จะคงไว้ใน Reduced Model ขั้นตอนต่อมาก็คือ การคำนวณค่าพารามิเตอร์ F, G, H และ L ซึ่งได้แสดงไว้แล้ว สำหรับขั้นตอนเหล่านี้ได้แสดงไว้ในรูปที่ 2



รูปที่ 2 แสดงขั้นตอนการคำนวณค่าพารามิเตอร์ของ Reduced Model



รูปที่ 2 (ต่อ) แสดงขั้นตอนการคำนวณค่าพารามิเตอร์ของ Reduced Model

3.7 เงื่อนไขสถานะเริ่มต้นของ Reduced Model

ในการแก้สมการเสตทสเปซเพื่อหาเวคเตอร์สถานะของระบบเชิงเส้นโดยทั่วไป แล้วจะหาได้จากสมการที่ (3.41) คือ

$$x(t) = \phi(t-t_o)x(t_o) + \int_{t_o}^{t} \phi(t-\tau)Bu(\tau)\,d\tau$$

เมื่อ ϕ คือ state transition matrix

ที่เวลาเริ่มต้น (initial time) $t_o = \phi$ สมการที่ (3.41) จะได้

$$x(t) = \phi(t)x(0) + \int_{t_o}^{t} \phi(t-\tau)Bu(\tau)\,d\tau \qquad (3.73)$$

จากสมการที่ (3.73) เราจะพบว่าการหาค่าเวคเตอร์สถานะจำเป็นต้องทราบค่าของเงื่อนไขสถานะเริ่มต้น (initial condition) ของ x(t) สำหรับการแก้สมการเพื่อหาค่าเวคเตอร์สถานะ x_ρ ของ Reduced Model ก็เช่นเดียวกัน เราจำเป็นต้องทราบค่าเงื่อนไขสถานะเริ่มต้นของ x_ρ ด้วย

สำหรับในหัวข้อต่อไปนี้ จะเสนอวิธีการหาค่าเงื่อนไขสถานะเริ่มต้น $x_\rho(0)$ ของ Reduced Model ว่าจะมีรูปแบบและมีความสัมพันธ์กับเงื่อนไขสถานะเริ่มต้น x(0) ของระบบเดิมอย่างไร

จากสมการที่ (3.58) เราใช้ค่า z_f และ x_ρ แทนการประมาณค่าของ z_2 และ x_1 ตามลำดับ เมื่อเราแทนค่า z_f และ x_ρ ลงในสมการที่ (3.58) จะได้ว่า

$$\begin{bmatrix} x_\rho \\ x_2 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ z_f \end{bmatrix} \qquad (3.74)$$

ถ้าให้ t = 0 จะแก้สมการ (3.74) เพื่อหาค่า $x_\rho(0)$ ได้คือ

$$x_{\rho}(0) = V_{11}z_1(0) + V_{12}z_{\xi}(0) \qquad (3.75)$$

เมื่อพิจารณาจากค่าเงื่อนไขเริ่มต้น $x_{\rho}(0)$ ในสมการที่ (3.75) จะพบว่าในการแก้สมการนี้เราจะต้องหาค่าเงื่อนไขสถานะเริ่มต้น $z_1(0)$ และ $z_{\xi}(0)$ ซึ่งจะหาได้จากสมการที่ (3.57) และ (3.62) ตามลำดับ

ถ้าให้ $t = 0$ ส่วนแรกของสมการที่ (3.57) จะได้เป็น

$$z_1(0) = U_{11}x_1(0) + U_{12}x_2(0) \qquad (3.76)$$

และสมการที่ (3.62) จะได้

$$z_{\xi}(0) = -\Lambda_2^{-1}\tilde{B}_2u(0) \qquad (3.77)$$

แทนค่า $z_1(0)$ และ $z_{\xi}(0)$ จากสมการที่ (3.76) และ (3.77) ลงในสมการที่ (3.75) จะได้

$$x_{\rho}(0) = V_{11}\left[U_{11}x_1(0)+U_{12}x_2(0)\right] - V_{12}\Lambda_2^{-1}\tilde{B}_2u(0) \qquad (3.78)$$

สมการที่ (3.78) คือเงื่อนไขสถานะเริ่มต้นของเวคเตอร์สถานะในตัวแบบของ Reduced Model จะเห็นได้ว่า $x_{\rho}(0)$ จะขึ้นกับเงื่อนไขสถานะเริ่มต้น $x_1(0)$ และ $x_2(0)$ ของตัวแบบในระบบเดิมด้วย

3.8 การวิเคราะห์ความผิดพลาด

จากทฤษฎีที่ใช้ในการสร้าง Reduced Model จะพบเงื่อนไขที่สำคัญประการหนึ่งคือ เอาท์พุตของระบบในตัวแบบที่สร้างขึ้นใหม่จะต้องใกล้เคียงกับเอาท์พุตของตัวแบบในระบบเดิมให้มากที่สุด ซึ่งเป็นที่แน่นอนว่าเอาท์พุตนี้อาจจะแตกต่างจากระบบเดิมอยู่บ้าง ในหัวข้อนี้ จะแสดงถึงการวิเคราะห์ความผิดพลาดที่เกิดขึ้นจากการลดทอนตัวแบบในวิธี

Singular Perturbation

การสร้าง Reduced Model มีการประมาณค่าที่สำคัญคือในสมการที่ (3.61) ได้มีการแทนค่า z_2 ด้วยค่า steady-state $\bar{z}_2$ และมีการแทนค่า $\bar{z}_2$ ด้วย z_f ตามสมการที่ (3.62) การแทนค่า z_2 ด้วย $\bar{z}_2$ นี้ เป็นรูปแบบในวิธี Singular Perturbation และในวิธีนี้เราจะต้องแสดงให้เห็นว่า ถ้า $u(t) \to \bar{u}$ ซึ่งเป็นค่าคงที่แล้ว $x_\rho(t) \to x_1(t)$ และ $\tilde{x}_2(t) \to x_2(t)$ เมื่อ $t \to \infty$

จากการวิเคราะห์ในวิธี Singular Perturbation เราจะพบว่าเวคเตอร์สถานะ x มีความสัมพันธ์กับเวคเตอร์สถานะ z คือ

$$x = Vz \tag{3.79}$$

เมื่อเราแบ่งเมตริกซ์ในสมการที่ (3.79) ให้สอดคล้องกันจะได้เป็น

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \tag{3.80}$$

ถ้าใช้การประมาณค่าให้เป็นไปตามสมการที่ (3.74) ซึ่งมีรูปแบบเป็น

$$\begin{bmatrix} x_\rho \\ x_2 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ z_f \end{bmatrix} \tag{3.81}$$

จากสมการที่ (3.80) และสมการที่ (3.81) เราจะได้

$$x_1 - x_{1f} = V_{12}(z_2 - z_{2f}) \tag{3.82}$$

$$x_2 - x_{2f} = V_{22}(z_2 - z_{2f}) \tag{3.83}$$

จากวิธี Singular Perturbation ที่ผ่านมา เราจะเห็นได้ว่ามีการแทนค่า z_2 ด้วย $\bar{z}_2$ และแทนค่า $\bar{z}_2$ ด้วย z_{2f} ดังนั้นในสมการที่ (3.82) และ (3.83) สามารถเขียนให้อยู่ในรูปแบบที่มีความถูกต้องมากขึ้นได้เป็น

$$x_1 - x_{1f} = V_{12}(z_2 - \bar{z}_2) - V_{12}(z_{2f} - \bar{z}_2) \tag{3.84}$$

$$x_2 - \tilde{x}_2 = V_{22}(z_2 - \bar{z}_2) - V_{22}(z_{2f} - \bar{z}_2) \tag{3.85}$$

สมการที่ (3.84) และ (3.85) จะแสดงถึงค่าความผิดพลาดของเวคเตอร์สถานะ x_1 และ x_2 ตามลำดับ

จะเห็นได้ว่า ค่าความผิดพลาดของเวคเตอร์สถานะในวิธี Singular Perturbation นี้ ขึ้นกับค่าความผิดพลาดที่เกิดจากการประมาณค่า $(z_2 - \bar{z}_2)$ และ $(z_{2f} - \bar{z}_2)$

จากสมการที่ (3.61) และ (3.62) จะได้ค่า $(z_{2f} - \bar{z}_2)$ เป็น

$$z_{2f} - \bar{z}_2 = -\Lambda_2^{-1}\tilde{B}_2(u - \bar{u}) \tag{3.86}$$

และจากเงื่อนไข steady-state ที่ว่าค่า $u(t)$ จะมีค่าเข้าสู่ $\bar{u}$ เมื่อ t มีค่าเข้าสู่อนันต์ ($u(t) \to \bar{u}$ เมื่อ $t \to \infty$) และพิจารณาจากสมการที่ (3.86) จะพบว่า $z_{2f}(t) \to \bar{z}_2$ เมื่อ $t \to \infty$

สำหรับกรณีของค่าความผิดพลาด $z_2 - \bar{z}_2$ ก็ใช้เงื่อนไข steady-state เช่นเดียวกับกรณีของ $z_{2f} - \bar{z}_2$ เราจะพบว่า $|[z_2(t) - z_2]_i| \to 0$ เมื่อ $i = 1, 2, \ldots, n-r$ จากนั้นเราสามารถสรุปได้ว่า $z_2(t) \to \bar{z}_2$ เมื่อ

$t \to \infty$ ดังนั้นเทอมที่อยู่ทางขวาของสมการที่ (3.84) และ (3.85) จะมีค่าเข้าสู่ศูนย์ เมื่อ t มีค่าเข้าสู่อนันต์ นั่นคือ ค่าความผิดพลาดของเวคเตอร์สถานะระหว่าง x_1 กับ x_ρ และ x_2 กับ $\tilde{x}_2$ ที่แสดงไว้ในสมการที่ (3.84) และ (3.85) ตามลำดับ จะมีเงื่อนไขที่สำคัญคือ ค่าความผิดพลาดนี้จะมีค่าเข้าสู่ศูนย์เมื่อ t มีค่ามาก

สำหรับค่าความผิดพลาดของเอาท์พุตจากระบบเดิม (y) และเอาท์พุตที่ได้จากการลดทอนตัวแบบ (y_ρ) แสดงได้เป็น

$$y - y_\rho = C_1(x_1 - x_\rho) + C_2(x_2 - \tilde{x}_2) \qquad (3.87)$$

จากเงื่อนไขของค่าความผิดพลาดของเวคเตอร์สถานะในสมการที่ (3.84) และ (3.85) ทำให้เราทราบว่า ค่าความผิดพลาดของเอาท์พุตในสมการที่ (3.87) มีค่าเข้าสู่ศูนย์เมื่อ t มีค่าเข้าสู่อนันต์ นั่นคือ ค่าเอาท์พุตของระบบที่ได้จากการลดทอนตัวแบบแล้ว (y_ρ) จะมีค่าใกล้เคียงกับเอาท์พุตของระบบเดิม (y) เมื่อเวลาผ่านไปนานๆ ซึ่งค่า y_ρ ที่มีค่าใกล้เคียงกับ y นี้ เป็นสิ่งที่เราต้องการและเป็นเงื่อนไขที่สำคัญที่สุดของ Reduced Model

บทที่ 4
การวิเคราะห์การลดทอนตัวแบบในโดเมนของความถี่

จากบทที่ 3 ที่ผ่านมา จะเห็นได้ว่าเราสามารถลดมิติของเมตริกซ์ที่ปรากฏในสมการเสตทสเปซได้เมื่อเราศึกษาระบบในโดเมนของเวลา แต่ในบางครั้ง การศึกษาระบบในโดเมนของความถี่ หรือการแสดงระบบในรูปของสมการ transfer function การสร้าง Reduced Model ตามวิธีการที่แสดงไว้ในบทที่ 3 จะกระทำได้ไม่สะดวกนัก ทำให้มีผู้เสนอวิธีการลดทอนตัวแบบในโดเมนของความถี่หลายวิธีด้วยกัน เช่น วิธีการประมาณค่าพลังงานอิมพัลซ์ [26,31] วิธีการประมาณค่าแบบ Chebyshev [5] วิธี Continued - Fraction Expansion [24,25,29] หรืออื่น ๆ ซึ่งส่วนใหญ่จะใช้ทฤษฎีการประมาณค่าทางคณิตศาสตร์

สำหรับบทนี้ จะเสนอวิธีการลดทอนตัวแบบในโดเมนของความถี่ โดยการนำวิธีการประมาณค่าแบบ Pade´ มาประยุกต์ใช้ ตลอดจนแสดงถึงการแก้ปัญหาเกี่ยวกับความมีเสถียรภาพ (stable) ของระบบ

4.1 ความหมายของ Reduced Model ในโดเมนของความถี่

จากบทที่ 2 เราพบว่าในการศึกษาระบบเชิงเส้นที่ไม่ขึ้นกับเวลาในโดเมนของความถี่ แสดงได้ด้วยสมการ transfer function คือ

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_o}{a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_o} \qquad (4.1)$$

เมื่อทำการลดทอนตัวแบบเพื่อสร้าง Reduced Model จะได้สมการ transfer function ของระบบใหม่ซึ่งมี Order น้อยกว่าระบบเดิมแทนด้วยสมการ คือ

$$R(s) = \frac{q_k s^k + q_{k-1} s^{k-1} + \ldots + q_1 s + q_o}{p_1 s^1 + p_{1-1} s^{1-1} + \ldots + p_1 s + p_o} \qquad (4.2)$$

โดยที่ $k < m$ และ $l < n$

สำหรับผลตอบสนองของระบบ Reduced Model ในโดเมนของความถี่ ก็มีเงื่อนไขเช่นเดียวกับ Reduced Model ในโดเมนของเวลา คือ ผลตอบสนองของระบบ Reduced Model นี้จะต้องมีค่าใกล้เคียงกับระบบเดิมให้มากที่สุด

เมื่อพิจารณา สมการที่ (4.1) และ (4.2) แล้วจะเห็นได้ว่า Reduced Model ในโดเมนของความถี่ คือ " การลดทอน Order ของสมการแสดง transfer function ของระบบนั่นเอง"

4.2 การประมาณค่าแบบ Pade´

จากความหมายของ Reduced Model ในโดเมนของความถี่ในหัวข้อ 4.1 ที่ผ่านมา เราจะเห็นได้ว่าปัญหาที่ติดตามมาคือ ค่าสัมประสิทธิ์ของสมการ transfer function ของ Reduced Model ควรเป็นอะไร และระบบที่ทำการลดทอนแล้วจะมีเสถียรภาพหรือไม่

ในหัวข้อนี้จะเสนอวิธีการคำนวณค่าสัมประสิทธิ์ของ Reduced Model โดยจะเสนอวิธีการประมาณค่าแบบ Pade´ มาช่วยในการแก้ปัญหา

พิจารณาฟังก์ชั่นที่อยู่ในรูปอนุกรมกำลัง

$$f(s) = c_o + c_1 s + c_2 s^2 + \ldots\ldots \qquad (4.3)$$

และฟังก์ชั่นที่มีรูปแบบเป็น $\left[Y_m(s) \,/\, U_n(s) \right]$ เมื่อ $Y_m(s)$ และ $U_n(s)$ เป็นสมการ Polynomials ของตัวแปร s ที่มี Order เท่ากับ m และ n ตามลำดับ ฟังก์ชั่น $\left[Y_m(.) \,/\, U_n(.) \right]$ จะเรียกว่าเป็นการประมาณค่าแบบ Pade´ ของ f(s) ก็ต่อเมื่อ ฟังก์ชั่นในรูปอนุกรมกำลังของ f(s) เป็นจำนวน (m+n) เทอมแรก และ $\left[Y_m(s) \,/\, U_n(s) \right]$ มีค่าเหมือนกัน (Identical) [9,18,36]

สำหรับฟังก์ชั่น f(s) ในสมการที่ (4.3) เป็นการประมาณค่าโดยใช้การประมาณค่าแบบ Pade´[18,34,36] โดยมีนิยามคือ

$$\frac{Y_m(s)}{U_n(s)} = \frac{a_o + a_1 s + a_2 s^2 + \ldots + a_m s^m}{b_o + b_1 s + b_2 s^2 + \ldots + b_m s^m} \qquad (4.4)$$

สมการที่ (4.3) จำนวน (m+n) เทอมแรก จะมีค่าเท่ากับสมการที่ (4.4) และจากความสัมพันธ์ระหว่างสมการที่ (4.3) และ (4.4) สามารถแทนได้ด้วยสมการ

$$\begin{aligned}
a_o &= b_o c_o \\
a_1 &= b_o c_1 + b_1 c_o \\
a_2 &= b_o c_2 + b_1 c_1 + b_2 c_o \\
&\vdots \\
a_m &= b_o c_m + b_1 c_{m-1} + \ldots + b_m c_o \\
&\vdots \\
0 &= b_o c_{m+1} + b_1 c_m + \ldots + b_{m+1} c_o \\
0 &= b_o c_{m+n} + b_1 c_{m+n-1} + \ldots + b_{n-1} c_{m+1} + c_m
\end{aligned} \qquad (4.5)$$

จะเห็นได้ว่าสมการที่ (4.4) อยู่ในรูปแบบของสมการ transfer function ค่าสัมประสิทธิ์ c_i (i = 0,1,2, ...) หาได้จากสมการที่ (4.5) ซึ่งสามารถเขียนให้อยู่ในรูปแบบเมตริกซ์ได้ คือ

$$\begin{bmatrix} c_{m+1} & c_m & \cdots\cdots\cdots & c_1 \\ c_{m+2} & c_{m+1} & \cdots\cdots\cdots & c_2 \\ c_{m+3} & c_{m+2} & c_{m+1} \cdots\cdots & c_3 \\ \vdots & \vdots & \vdots \quad \vdots & \vdots \\ \vdots & \vdots & \vdots \quad \vdots & \vdots \\ c_{m+n} & c_{m+n-1} & \cdots\ c_{m+2} \quad c_{m+1} \end{bmatrix} \begin{bmatrix} b_o \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} -c_o \\ -c_1 \\ -c_2 \\ \vdots \\ \vdots \\ -c_m \end{bmatrix} \qquad (4.6)$$

$$\begin{bmatrix} a_o \\ a_1 \\ a_2 \\ \vdots \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} c_o & 0 & \cdots\cdots\cdots & 0 \\ c_1 & c_o & 0 \cdots\cdots & 0 \\ c_2 & c_1 & c_o \quad 0 \cdots & 0 \\ \vdots & \vdots & \vdots \cdots 0 \cdots & \vdots \\ \vdots & \vdots & \vdots \cdots & \vdots \\ c_m & c_{m-1} & \cdots\cdots\ c_1 \quad c_o \end{bmatrix} \begin{bmatrix} b_o \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n-1} \end{bmatrix} \qquad (4.7)$$

โดยที่สมการที่ (4.7) แทนชุดของสมการที่ (4.5) จำนวน (m+1) ส่วนสมการที่ (4.6) จะแทนชุดของสมการ (4.5) ที่เหลือ

สมการที่ (4.6) และ (4.7) เป็นรูปแบบใหม่ของสมการที่ (4.5) และสำหรับรูปแบบของ transfer function โดยทั่วไปมักจะให้ $b_n = b_{m+1} = 1$ สมการที่ (4.6) และ (4.7) สามารถเขียนให้ง่ายขึ้นโดยเขียนให้อยู่ในรูปแบบของพีชคณิตเมตริกซ์ได้เป็น

$$C_1 b = -c \qquad (4.8)$$

$$a = C_2 b \qquad (4.9)$$

สมการที่ (4.8) และ (4.9) แทนสมการที่ (4.6) และ (4.7) ตามลำดับ โดยที่ C_1 เป็นเมตริกซ์มีมิติเท่ากับ (n*n) ส่วน C_2 มีมิติเท่ากับ (m+1)*(m+1) a, b,และ c เป็นเวคเตอร์สดมภ์มีขนาดเท่ากับ (m+1), (n), และ (m+1) ตามลำดับ

จากแนวความคิดของทฤษฎีการประมาณค่าแบบ Pade' นี้จะเห็นได้ว่า ถ้าเรามีสมการ transfer function มีรูปแบบตามสมการที่ (4.4) แล้วจัดการให้อยู่ในรูปแบบของสมการที่ (4.3) เมื่อเราเลือกจำนวน Order ที่จะคงไว้ใน Reduced Model แล้ว จึงคำนวณค่าสัมประสิทธิ์ของ Reduced Model ได้จากชุดสมการที่ (4.5)

ถึงแม้ว่าแนวความคิดนี้จะมีผู้นำไปประยุกต์ใช้ในการออกแบบ Reduced Model ในโดเมนของความถี่กันอย่างกว้างขวาง [7,14,16,34,35] แต่วิธีนี้ก็ยังมีข้อบกพร่องคล้ายกับแนวความคิดในการเลือกสถานะที่จะคงไว้ในโดเมนของเวลาซึ่งได้แสดงไว้ในหัวข้อที่ (3.5.1) จากหัวข้อที่ (3.5.1) เราได้พิสูจน์จนได้สมการที่ (3.44) ซึ่งนำมาใช้ในการพิจารณาเลือกสถานะโดยวิธีการนั้นไม่ได้นำเมตริกซ์ B และ C ซึ่งปรากฏในสมการเสตทสเปซมาพิจารณาทำให้ได้ผลตอบสนองมีความผิดพลาดอยู่พอสมควร สำหรับวิธีการประมาณค่าแบบ Pade' ก็เช่นกัน จะเห็นได้ว่ามีการนำค่าสัมประสิทธิ์ของระบบเดิมมาช่วยในการคำนวณค่าสัมประสิทธิ์ของ Reduced Model แต่ไม่มีการนำเอาค่าโพล (pole) ของระบบมาพิจารณาเลย ทำให้วิธีการนี้ได้ผลไม่ดีนัก

4.3 ความมีเสถียรภาพของ Reduced Model

ในหัวข้อที่ (4.2) จะเห็นได้ว่าวิธีการประมาณค่าแบบ Pade' สามารถนำมาประยุกต์ใช้ในการคำนวณค่าสัมประสิทธิ์ของสมการ transfer function ที่ปรากฏใน Reduced Model ได้ แต่ก็ยังมีข้อเสียอยู่บ้าง คือ นอกจากจะไม่ได้นำค่าโพลของระบบเดิมมาพิจารณาแล้ว บางครั้งตัวแบบที่ได้จากการลดทอนแล้วยังไม่มีเสถียรภาพอีกด้วย แม้ว่าระบบเดิมจะเป็นระบบที่มีเสถียรภาพก็ตาม

มีผู้เสนอวิธีการประยุกต์ใช้การประมาณค่าแบบ Pade' ร่วมกับทฤษฎีระบบควบคุมหลายวิธีด้วยกัน สำหรับในหัวข้อนี้ จะเสนอวิธีการประมาณค่าแบบ Pade' มาประยุกต์ใช้ร่วมกับวิธี Stability - Equation [1,7,19,28] และ วิธี Fitting Time-Moments [18,27,34,35] จากนั้นจึงจะแสดงขั้นตอนการคำนวณ

(Algorithms) เพื่อพัฒนาเป็นโปรแกรมสำเร็จรูปต่อไป

4.3.1 การหาระบบที่เสถียรโดยใช้วิธี Stability-Equation

สำหรับหัวข้อนี้จะเสนอวิธีการวิเคราะห์ระบบ Reduced Model โดยการนำวิธีการประมาณค่าแบบ Pade' มาประยุกต์ใช้ร่วมกับวิธี Stability-Equation วิธีการนี้เป็นวิธีที่นิยมกันมาก มีบทความเสนอถึงการนำแนวความคิดนี้ไปประยุกต์ใช้หลายบทความด้วยกัน [3,4,7,28] วิธีการนี้จะทำให้ได้ตัวแบบหลังจากการลด Order แล้ว เป็นตัวแบบที่มีเสถียรภาพ

พิจารณาระบบที่มีสมการ Transfer Function เป็น

$$F(s) = \frac{b_o + b_1 s + b_2 s^2 + \ldots + b_{n-1} s^{n-1}}{a_o + a_1 s + a_2 s^2 + \ldots + a_{n-1} s^{n-1} + a_n s^n} \qquad (4.10)$$

สมการที่ (4.10) สามารถทำให้อยู่ในรูปของอนุกรมกำลังของ s ได้คือ [7,18]

$$F(s) = \sum_{n=1}^{\infty} M_i s^{-i-1}$$

$$= \sum_{n=1}^{\infty} T_i s^i \qquad (4.11)$$

เมื่อ M_i คือ Markov parameter ตัวที่ i ของระบบ

และ T_i คือ time-moment ตัวที่ i ของระบบ

สำหรับระบบที่เสถียร สมการเสถียรภาพ (Stability - Equation) ของตัวหาร (Denominator) ในสมการที่ (4.10) เขียนได้เป็น [7, 28]

$$F_{De}(s) = \sum_{i=0,2,4}^{n} a_i s^i = a_o \prod_{i=1}^{n/2} (1 + s^2/z_i) \qquad (4.12a)$$

$$F_{Do}(s) = \sum_{i=1,3,5}^{n} a_i s^i = a_1 s \prod_{i=1}^{(n-1)/2} (1 + s^2/p_i) \qquad (4.12b)$$

ค่า z_1 และ p_1 ที่ปรากฏในสมการที่ (4.12a) และ (4.12b) คือค่าโพลของระบบโดยที่มีความสัมพันธ์คือ $z_1 < p_1 < z_2 < p_2 < z_3 < p_3 < \ldots$

หลังจากเราลดค่า Order ของสมการ transfer function ให้เหลือ Order เท่ากับ k โดยการตัดโพลที่มีค่ามาก ๆ ทิ้งไปจนกระทั่งเหลือจำนวน Order ตามที่ต้องการทำให้เขียนสมการที่ (4.12a) และ (4.12b) ในรูปแบบใหม่ได้คือ

$$F_{Dk}(s) = F_{De}(s) + F_{Do}(s) = \sum_{j=0}^{k} e_j s^j \qquad (4.13)$$

เมื่อ $F_{De}(s) = a_o \prod_{i=1}^{k/2} (1 + s^2/z_i)$

และ $F_{Do}(s) = a_1 s \prod_{i=1}^{(k-1)/2} (1 + s^2/p_i)$

ถ้าสมการ transfer function ของ Reduced Model มีรูปแบบเป็น

$$F_k(s) = \frac{d_o + d_1 s + d_2 s^2 + \ldots + d_{k-1} s^{k-1}}{e_o + e_1 s + e_2 s^2 + \ldots + e_{k-1} s^{k-1} + e_k s^k} \qquad (4.14)$$

และค่า time-moments ตัวที่ 1 ถึง α ค่า Markov parameter ตัวที่ 1 ถึง β หาได้จากสมการ transfer function ของระบบต้นแบบ (Original system)และระบบของ Reduced Model โดยที่ผลรวมของ α และ β เท่ากับ k

$$\begin{aligned}
d_o &= e_o T_o \\
d_1 &= e_o T_1 + e_1 T_o \\
&\vdots \quad \vdots \quad \vdots \\
d_{\alpha-1} &= e_o T_{\alpha-1} + e_1 T_{\alpha-2} + \ldots + e_{\alpha-2} T_1 + e_{\alpha-1} T_o \qquad (4.15) \\
d_{k-\beta} &= e_k M_{\beta-1} + e_{k-1} M_{\beta-2} + \ldots + e_{\beta-1} M_1 + e_\beta M_o \\
&\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots
\end{aligned}$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$d_{k-2} = e_k M_1 + e_{k-1} M_0$$

$$d_{k-1} = e_k M_0$$

เมื่อคำนวณค่า d_1 แล้วนำไปแทนค่าในสมการที่ (4.14) ก็จะได้สมการ transfer function ของ Reduced Model ตามต้องการ

จะเห็นได้ว่าวิธีการ Stability - Equation นี้ เป็นวิธีการเลือกสถานะที่จะคงไว้ใน Reduced Model โดยการพิจารณาจากค่าโพลของระบบและจะตัดโพลที่มีความสำคัญไม่มากทิ้งไป แล้วจึงคำนวณค่าสัมประสิทธิ์ของ Reduced Model โดยการนำวิธีการประมาณค่าแบบ Pade´ มาประยุกต์ใช้

วิธีการเลือกสถานะที่จะคงไว้ โดยการพิจารณาจากขนาดของโพล ในวิธี Stability - Equation นี้ คล้ายกับวิธีการเลือกสถานะที่จะคงไว้โดยการพิจารณาจากค่าไอเก็น หรือค่าพลังงานของระบบในโดเมนของเวลา [1,12] โดยในโดเมนของเวลาจะต้องพิจารณาค่าไอเก็นซึ่งสามารถหาได้จากสมการ Characteristic ซึ่งมีรูปแบบเป็น

$$\lambda^n + a_{n-1}\lambda^{n-1} + \ldots + a_1\lambda + a_0 = 0 \qquad (4.16)$$

สำหรับในโดเมนของความถี่ที่มีสมการ transfer function ของระบบมีรูปแบบตามสมการที่ (4.10) ค่าโพลของระบบซึ่งจะนำมาพิจารณาจะหาได้จากสมการ Characteristic Polynomial คือ

$$s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0 = 0 \qquad (4.17)$$

จะเห็นได้ว่าวิธีการเลือกสถานะที่จะคงไว้ในโดเมนของเวลาและในโดเมนของความถี่มีลักษณะคล้ายกันมาก ต่อมาวิธีการ Stability - Equation นี้ได้รับการพัฒนาเป็นรูปแบบต่าง ๆ จำนวนมาก [1,3,5,28]

4.3.2 การสร้างตัวแบบของ Reduced Model ที่เสถียรโดยวิธี Fitting Time - Moments

จากหัวข้อที่ (4.3.1) ได้เสนอวิธีการหาระบบ Reduced Model ที่เสถียรโดยการประยุกต์วิธี Stability - Equation จะเห็นได้ว่าเป็นวิธีที่ค่อนข้างจะสมบูรณ์วิธีหนึ่ง แต่เมื่อนำไปพัฒนาเป็นโปรแกรมสำเร็จรูปจะเห็นได้ว่ามีขั้นตอนการคำนวณที่ยุ่งยากอยู่บ้าง คือการคำนวณค่าพารามิเตอร์ของ Markov และค่า time - moment ของระบบ ขั้นตอนที่ไม่สะดวกอีกก็คือ การกำหนดค่าโพลที่จะใช้ในสมการที่ (4.12a) และ (4.12b) ตลอดจนการกำหนดค่า α และ β ที่จะนำไปใช้ในการคำนวณค่า d_1 ในสมการที่ (4.15) เมื่อพัฒนาเป็นโปรแกรมแล้ว จะเห็นว่าในส่วนนี้ผู้ใช้งานจะต้องทำเองทั้งหมด ถึงแม้จะสามารถออกแบบขั้นตอนการคำนวณและโปรแกรมให้ทำงานได้ตามวิธี Stability - Equation แต่การใช้งานก็จะไม่สะดวกนัก

ในหัวข้อนี้จะเสนอวิธีการสร้างตัวแบบของ Reduced Model โดยวิธี Fitting Time - Moments ซึ่งเป็นวิธีที่สะดวกเข้าใจง่ายและเหมาะสมที่จะนำมาพัฒนาขั้นตอนการคำนวณเพื่อทำเป็นโปรแกรมช่วยคำนวณ วิธีการนี้มีพื้นฐานอยู่ที่การกำหนดค่าเริ่มต้น (initial) ของค่า time - moments ของระบบนอกจากนี้ยังสามารถแก้ไขปัญหาการไม่มีเสถียรภาพจากการสร้าง Reduced Model โดยการประมาณค่าแบบ Pade´ ได้อีกด้วย

กำหนดให้ผลตอบสนองอิมพัลซ์ (impluse response) $g(t)$ ของระบบที่เสถียรในโดเมนของความถี่เป็น

$$G(s) = \int_0^\infty g(t)e^{-st}\,dt \qquad (4.18)$$

เมื่อกระจาย e^{-st} ให้อยู่ในรูปของอนุกรมกำลัง สมการที่ (4.18) จะมีรูปแบบเป็น

$$G(s) = \int_0^\infty g(t)\left\{1 - st + \frac{(st)^2}{2!} - \frac{(st)^3}{3!} + \ldots\right\} dt$$

$$G(s) = \int_o^\infty g(t)dt - s\int_o^\infty tg(t)dt + s^2\int_o^\infty \frac{t^2}{2!} g(t)dt - \ldots \quad (4.19)$$

จากสมการที่ (4.19) สามารถเขียนให้อยู่ในรูปแบบใหม่ได้คือ

$$G(s) = c_o + c_1 s + c_2 s^2 + \ldots \quad (4.20)$$

สมการที่ (4.20) ที่ได้จากสมการที่ (4.19) นี้ ก็คือสมการที่มีรูปแบบเช่นเดียวกับสมการที่ (4.3) และ (4.11) นั่นเอง จากสมการที่ (4.20) นี้จะเห็นได้ว่าเราสามารถนิยามค่า c_i (i = 0, 1, 2,...) ของสมการที่ (4.20) ได้เป็น

$$c_i = \left[\frac{(-1)^i}{i!}\right] \int_o^\infty t^i g(t)\, dt \quad ; i=0, 1, 2, \ldots \quad (4.21)$$

หรือ $$c_i = \left[\frac{(-1)^i}{i!}\right] M_i \quad (4.22)$$

เมื่อ M_i คือ "ค่า time - moment ตัวที่ i ของ g(t)" [18,21,34]

จะเห็นว่าค่า time - moment ของระบบจะเป็นสัดส่วนกับอนุกรมกำลังของ G(s) ในวิธีการ time - moments นั้น สมการ transfer function ของ Reduced Model [R(s)] ก็คือการเลือกค่า moments ของ G(s) นั่นเอง

ถ้าให้ค่าสมการ transfer function ของระบบต้นแบบ มีรูปแบบเป็น

$$G(s) = \frac{d_o + d_1 s + d_2 s^2 + \ldots + d_{n-1} s^{n-1}}{e_o + e_1 s + e_2 s^2 + \ldots + e_{n-1} s^{n-1} + s^n} \quad (4.23)$$

โดยที่เมื่อเราลด Order ของสมการที่ (4.23) จะได้สมการ transfer function ของ Reduced Model มีรูปแบบเป็น

$$R(s) = \frac{a_o + a_1 s + a_2 s^2 + \ldots + a_{k-1} s^{k-1}}{b_o + b_1 s + b_2 s^2 + \ldots + b_{k-1} s^{k-1} + s^k} \qquad (4.24)$$

โดยที่ k มีค่าน้อยกว่า n

ถ้ากระจาย G(s) ให้อยู่ในรูปของอนุกรมกำลังตามสมการที่ (4.20) คำนวณค่า c_i (i = 0, 1, 2, ...) จากสมการที่ (4.21) หรือ (4.22) จากนั้นใช้การประมาณค่าแบบ Pade´ ที่เสนอไว้ในหัวข้อที่ 4.2 เราจะสามารถแก้ปัญหาเกี่ยวกับการหาค่าสัมประสิทธิ์ในสมการ transfer function ของ Reduced Model ได้โดยคำนวณค่าสัมประสิทธิ์ a_i (i=0, 1, 2,...) และ b_i (i= 0, 1, 2,...) ของ R(s) ได้จาก

$$\begin{aligned}
a_o &= b_o c_o \\
a_1 &= b_o c_1 + b_1 c_o \\
&\vdots \\
0 &= b_o c_{2k-2} + b_1 c_{2k-3} + \ldots + c_{k-1} \\
0 &= b_o c_{2k-1} + b_1 c_{2k-2} + \ldots + c_k
\end{aligned} \qquad (4.25)$$

เมื่อพิจารณาสมการที่ (4.20),(4.22) และ (4.25) จะเห็นได้ว่า ค่า time - moments ชุดแรกจำนวน 2k เทอม ของ R(s) และ G(s) มีค่าเท่ากัน

สำหรับเซ็ทของสมการที่ (4.25) นั้น พบว่ามีจำนวนทั้งสิ้น 2k สมการ โดยจำนวน k สมการแรกเราจะติดค่าตัวแปร a_i และ b_i อยู่ ดังนั้นการแก้สมการที่สะดวกจึงควรใช้ชุดสมการจำนวน k สมการท้ายคำนวณค่า b_i (i= 0, 1, 2,..., k-1) จากนั้นจึงใช้สมการจำนวน k สมการแรกคำนวณค่า a_i (i= 0, 1, 2,..., k-1) เราก็จะได้ค่าสัมประสิทธิ์ของสมการ transfer function ตามต้องการ

อย่างไรก็ตาม มักพบอยู่เสมอว่าวิธีการนี้อาจจะยังคงให้ระบบ Reduced Model ที่ไม่เสถียร สำหรับการแก้ปัญหานี้ มีนักวิจัยบางท่านเสนอว่า จะต้องมีการแก้ไขชุด

ของสมการที่ (4.25) บ้างเล็กน้อย [24,29,34,35] โดยเริ่มจากการคำนวณค่าโพลของระบบต้นแบบจากนั้นจึงเลือกค่าโพลที่มีขนาดเล็กที่สุดที่จะคงไว้ใน Reduced Model

ถ้าตัวแบบของ Reduced Model ที่ได้นี้ยังไม่เสถียรอีกให้เปลี่ยนมาใช้ค่าโพลที่จะคงไว้เป็นโพลที่มีขนาดใหญ่ที่สุดแทน เมื่อเลือกค่าโพลได้แล้ว สมการสุดท้ายในเซ็ทของสมการที่ (4.25) ให้แทนด้วยสมการ

$$0 = b_o - b_1 s_1 + b_2 s_1^2 + \ldots (-1)^k s_1^k \qquad (4.26)$$

เมื่อ s_1 คือโพลที่จะคงไว้ในสมการ transfer function R(s) ของ Reduced Model

4.4 การพัฒนาขั้นตอนการคำนวณวิธี Fitting Time - Moments

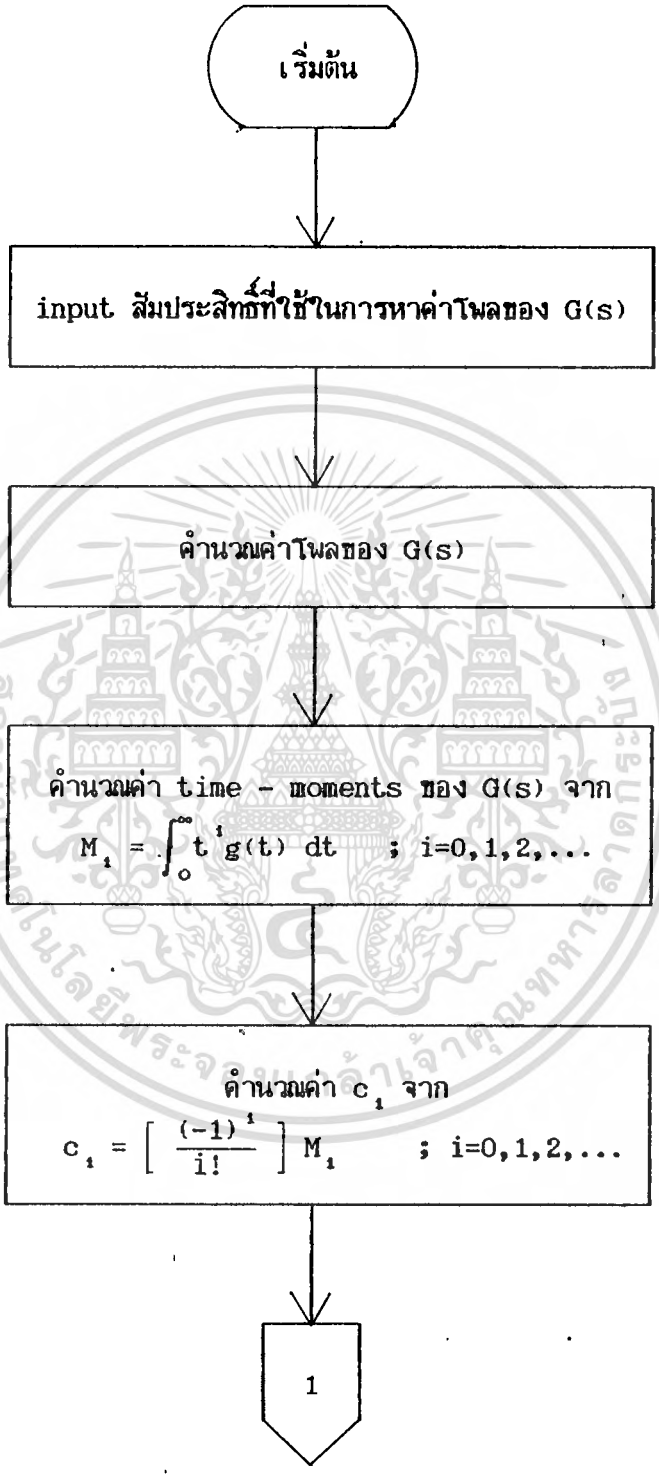
จากหัวข้อที่แล้วเราได้เสนอการแก้ปัญหาเกี่ยวกับความมีเสถียรภาพของระบบ Reduced Model โดยเสนอวิธี Stability - Equation และวิธี Fitting Time - Moments ซึ่งถึงแม้ว่าวิธี Stability - Equation จะเป็นวิธีการที่ค่อนข้างสมบูรณ์และเป็นที่นิยมมากมีงานวิจัยหลายชิ้นที่เสนอถึงการพัฒนาและการประยุกต์ใช้ แต่เราจะพบว่าวิธี Fitting Time - Moments เป็นวิธีที่ง่ายต่อการพัฒนาโปรแกรมมากกว่า

ในหัวข้อนี้จะเสนอวิธีการพัฒนาขั้นตอนการคำนวณ (Algorithms) ของวิธี Fitting Time - Moments เพื่อที่จะได้นำไปสร้างเป็นโปรแกรมต่อไป

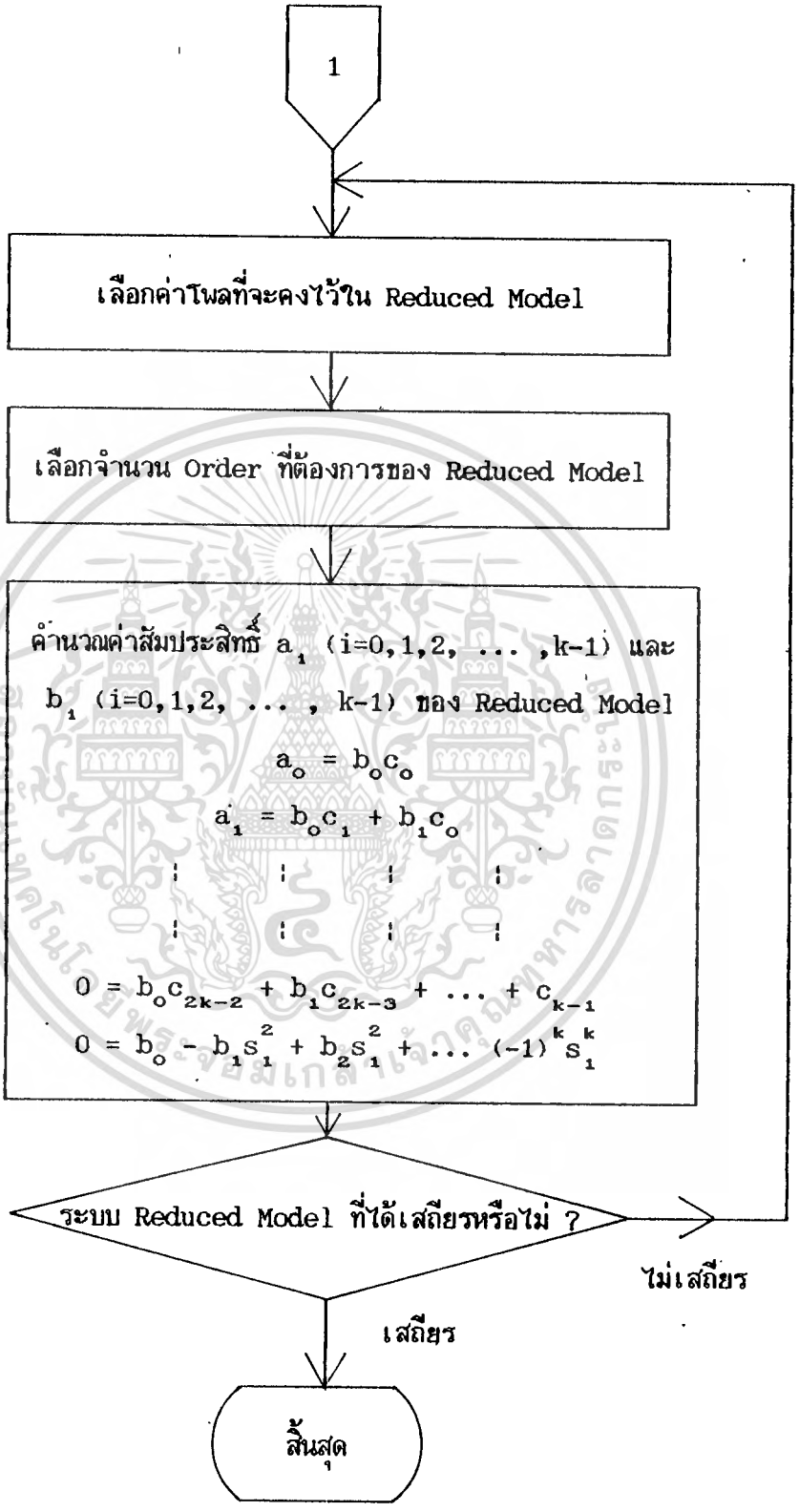
4.4.1 ขั้นตอนการคำนวณจากวิธี Fitting Time - Moments

สำหรับหัวข้อนี้จะแสดงขั้นตอนการคำนวณค่าสัมประสิทธิ์ของสมการ transfer function ใน Reduced Model จากวิธี Fitting Time - Moments ที่แสดงไว้ในหัวข้อที่ 4.3.2

เมื่อพิจารณาวิธีการที่เสนอไว้ในหัวข้อที่ 4.3.2 ถ้า G(s) มีรูปแบบเป็นไปตามสมการที่ (4.23) และ R(s) มีรูปแบบตามสมการที่ (4.24). จะแสดงเป็นขั้นตอนการคำนวณได้ตามรูปที่ 3



รูปที่ 3 ขั้นตอนการคำนวณในวิธี Fitting Time - Moments



รูปที่ 3 (ต่อ) ขั้นตอนการคำนวณในวิธี Fitting Time - Moments

4.4.2 การปรับปรุงขั้นตอนการคำนวณจากวิธี Fitting Time - Moments โดยใช้วิธีการประมาณค่าแบบ Pade´

จากขั้นตอนการคำนวณหาค่าสัมประสิทธิ์ของ Reduced Model ที่เสนอในหัวข้อที่แล้ว จะพบว่ามีข้อยุ่งยากอยู่ประการหนึ่งคือ การคำนวณค่า time - moments M_1 ซึ่งเป็นสมการอินทิเกรต สมการที่อยู่ในลักษณะนี้เมื่อนำไปพัฒนาเป็นโปรแกรม จะพบว่าเราต้องนิยามฟังก์ชั่นที่อินทิเกรตทุกครั้งที่มีการเปลี่ยนแปลงรูปแบบฟังก์ชั่น ทำให้การใช้โปรแกรมไม่สะดวก

สำหรับวิทยานิพนธ์นี้จะเสนอวิธีการคำนวณค่า c_1 โดยไม่ต้องคำนวณ M_1 ซึ่งเป็นสมการอินทิเกรต โดยนำวิธีการประมาณค่าแบบ Pade´ ที่ได้เสนอไว้ในหัวข้อที่ 4.2 มาประยุกต์ใช้

จากสมการ transfer function G(s) มีรูปแบบตามสมการที่ (4.23) คือ

$$G(s) = \frac{d_o + d_1 s + d_2 s^2 + \ldots d_{n-1} s^{n-1}}{e_o + e_1 s + e_2 s^2 + \ldots e_{n-1} s^{n-1} + s^n}$$

จากหัวข้อที่ (4.3.2) เราพบว่าสามารถเขียนให้อยู่ในรูปอนุกรมกำลังได้คือ

$$G(s) = \sum_{i=0}^{\infty} c_i s^i \qquad (4.27)$$

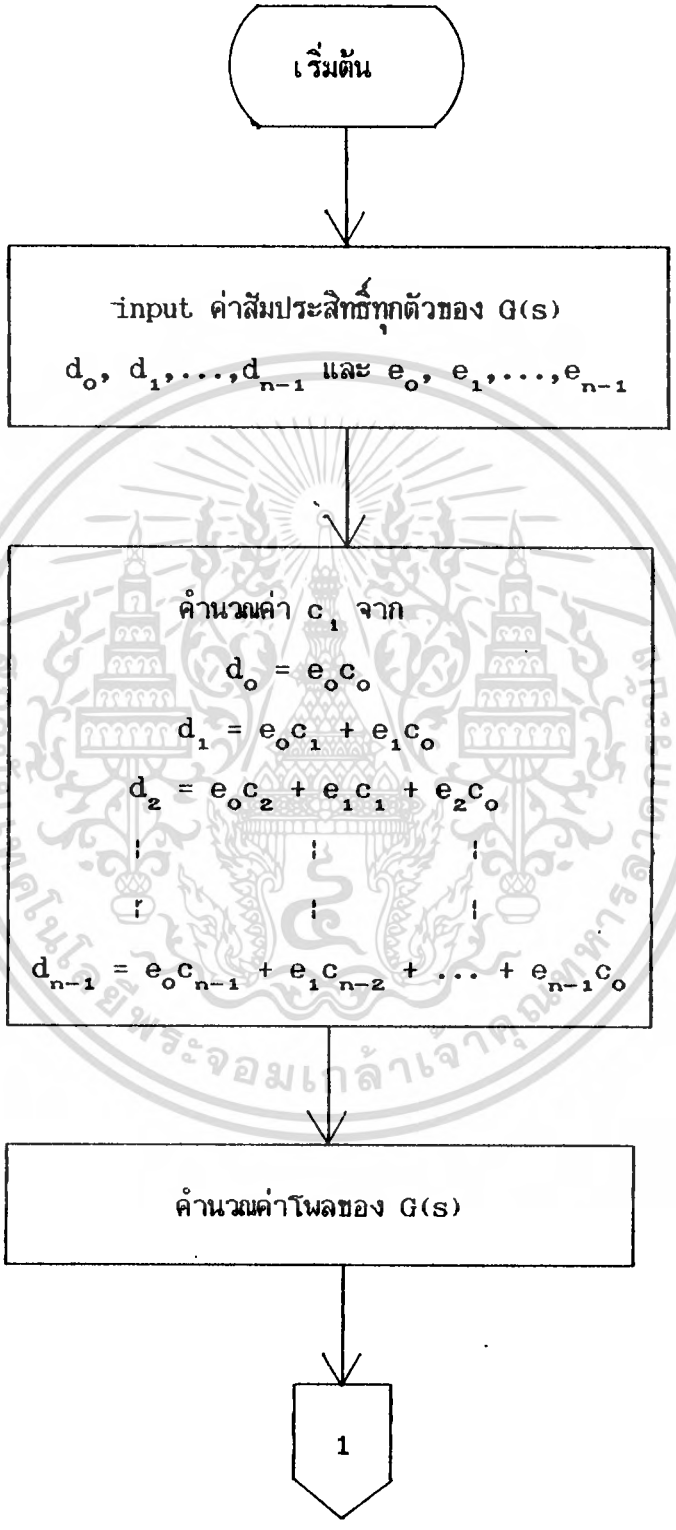
ค่า c_i หาได้จากสมการที่ (4.21) และ (4.22) เราจะเห็นว่าค่า G(s) ในสมการที่ (4.23) และ (4.27) นี้มีรูปแบบเช่นเดียวกับสมการที่ (4.4) และ (4.3) ตามลำดับ ดังนั้นเมื่อใช้การประมาณค่าแบบ Pade´ เราจะสามารถหาค่า c_i ได้จากชุดของสมการที่ (4.5) และจากการพิจารณาพบว่าเลือกใช้เพียงสมการจำนวน (m+1) สมการแรกก็จะสามารถคำนวณค่า c_i ได้ นั่นคือ

$$
\begin{aligned}
d_o &= e_o c_o \\
d_1 &= e_o c_1 + e_1 c_o \\
d_2 &= e_o c_2 + e_1 c_1 + e_2 c_o \\
&\vdots \qquad\quad \vdots \qquad\quad \vdots \\
&\vdots \qquad\quad \vdots \qquad\quad \vdots \\
d_{n-1} &= e_o c_{n-1} + e_1 c_{n-2} + \ldots + e_{n-1} c_o
\end{aligned}
\tag{4.28}
$$

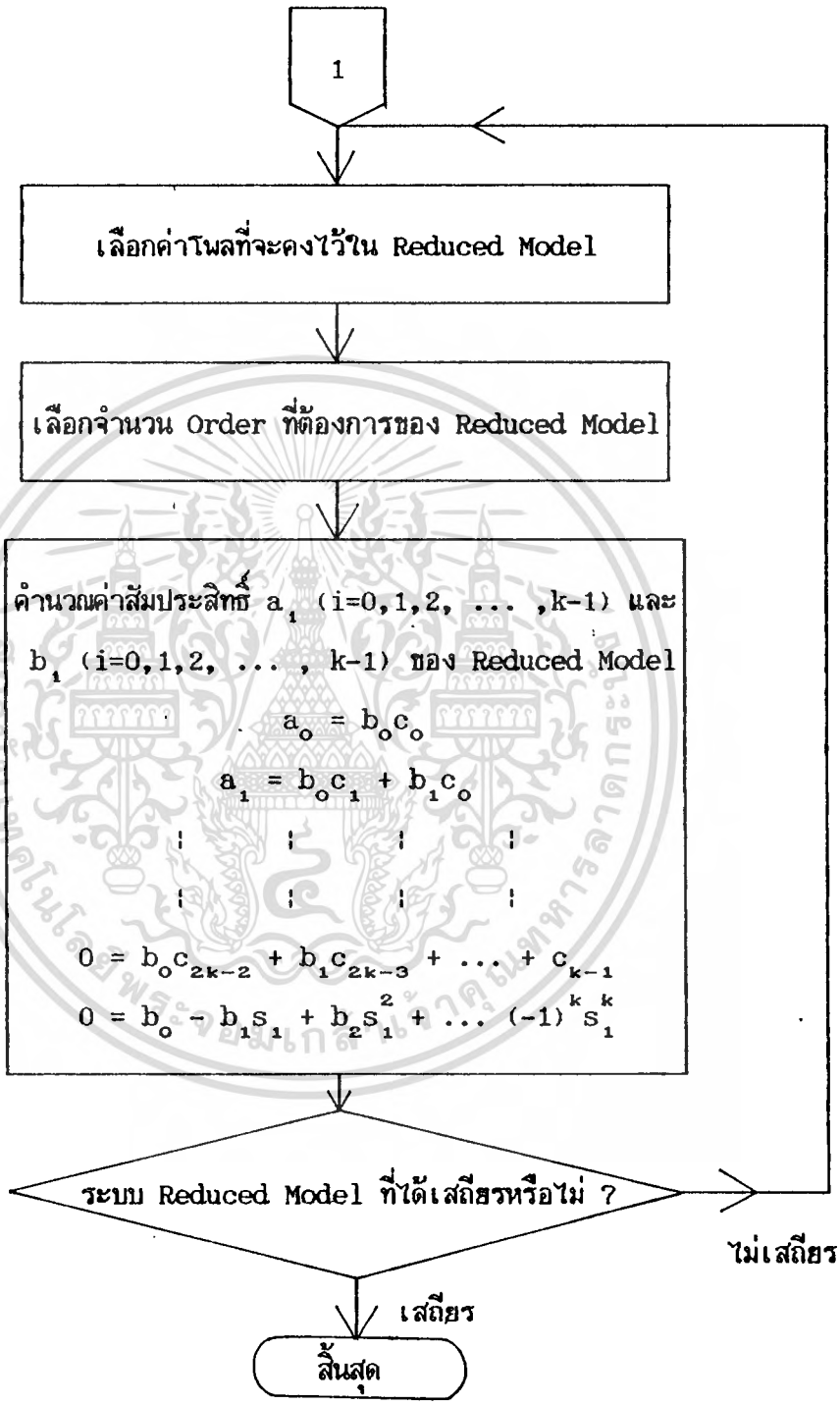
สมการที่ (4.28) เขียนให้อยู่ในรูปเมตริกซ์ได้คือ

$$
\begin{bmatrix} d_o \\ d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \end{bmatrix}
=
\begin{bmatrix}
e_o & 0 & \cdots & \cdots & 0 \\
e_1 & e_o & 0 & \cdots & 0 \\
e_2 & e_1 & e_o & \cdots & 0 \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
e_{n-1} & e_{n-2} & \cdots & e_1 & e_o
\end{bmatrix}
\begin{bmatrix} c_o \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_{n-1} \end{bmatrix}
\tag{4.29}
$$

จะเห็นได้ว่าเมื่อเราเปลี่ยนวิธีการคำนวณค่า c_1 จะทำให้ขั้นตอนการคำนวณในรูปที่ 3 จากหัวข้อที่ 4.4.1 เปลี่ยนแปลงไปตามรูปที่ 4



รูปที่ 4 ขั้นตอนการคำนวณในวิธี Fitting Time - Moments เมื่อใช้การประมาณค่าแบบ Pade' ช่วยในการหาค่าโมเมนต์ c_1



รูปที่ 4 (ต่อ) ขั้นตอนการคำนวณในวิธี Fitting Time - Moments เมื่อใช้การประมาณค่าแบบ Pade ช่วยในการหาค่าโมเมนต์ c_i

จากขั้นตอนการคำนวณที่แสดงไว้ในรูปที่ 4 จะเห็นว่าสะดวกต่อการพัฒนาเป็นโปรแกรมมาก

อย่างไรก็ตาม วิธีการนี้มีข้อจำกัดอยู่บ้างเกี่ยวกับการเลือกจำนวน Order ที่จะคงไว้ใน Reduced Model นั่นคือ ถ้าระบบต้นแบบมี Order เท่ากับ n และถ้าเลือกจำนวน Order ที่จะคงไว้ใน Reduced Model โดยให้มี Order เท่ากับ k แล้ว ค่า n และ k มีความสัมพันธ์กันคือ 2k-1 จะต้องน้อยกว่าหรือเท่ากับ n ถ้าหากไม่เป็นไปตามเงื่อนไขนี้แล้ว จะพบว่า สมการที่ (4.25) ตัวไม่ทราบค่า(unknow) c_i มากกว่าจำนวนสมการทำให้ไม่สามารถแก้สมการเชิงเส้นในการหาค่า c_i ได้

บทที่ 5

การทดสอบระบบ

ในบทต่าง ๆ ที่ผ่านมาของวิทยานิพนธ์ฉบับนี้ จะเห็นได้ว่าเราสามารถลดทอนตัวแบบของระบบได้โดยที่ในบทที่ 3 ได้แสดงถึงการสร้าง Model Reduction ในโดเมนของเวลา ส่วนในบทที่ 4 นั้นได้แสดงถึงการสร้าง Model Reduction ในโดเมนของความถี่ ซึ่งก็คือ การลดค่า Order ของ transfer function ของระบบนั่นเอง

ในการสร้าง Reduced Model ในโดเมนของเวลาและโดเมนของความถี่นั้น มีปัญหาที่สำคัญอีกประการหนึ่ง คือ เราจะทราบได้อย่างไรว่า ตัวแบบที่สร้างขึ้นใหม่นี้จะใช้แทนระบบเดิมได้ จากเงื่อนไขของ Reduced Model ที่ว่า เอาท์พุตของตัวแบบ Reduced Model และระบบเดิมจะต้องใกล้เคียงกันมากที่สุด ทำให้เราทราบว่า เราจะต้องศึกษาผลตอบสนองของระบบทั้งสองว่าใกล้เคียงกันมากน้อยเพียงใดจึงจะสามารถสรุปได้

5.1 การทดสอบระบบในโดเมนของเวลา

จากบทที่ 3 เราพบว่าในการสร้าง Model Reduction ในโดเมนของเวลาได้แบ่งการแก้ปัญหาออกเป็นสองส่วน คือ การแก้ปัญหาเกี่ยวกับการเลือกสถานะที่จะคงไว้และการคำนวณค่าพารามิเตอร์ของระบบ โดยการเลือกสถานะที่จะคงไว้จะใช้การพิจารณาค่าไอเก็นและพลังงานของระบบ ส่วนการคำนวณค่าพารามิเตอร์จะนำวิธี Singular Perturbation มาประยุกต์ใช้ ในหัวข้อนี้จะเสนอตัวอย่างบางระบบที่นำมาทดสอบในโดเมนของเวลา เพื่อเป็นการแสดงว่า ตัวแบบที่สร้างขึ้นใหม่นี้ใช้แทนระบบเดิมได้ดีเพียงใด

ตัวอย่าง 5.1.1

สำหรับตัวอย่างที่ 5.1.1 นี้เพื่อให้เห็นภาพพจน์ชัดเจนจะใช้ระบบทดสอบที่มีขนาดเล็ก

จากสมการเสตทสเปซที่แสดงไว้ในสมการที่ (3.1) และ (3.2)

กำหนดให้

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \end{bmatrix} u$$

$$y = \begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

เมื่อคำนวณค่าไอเก็นของเมตริกซ์ A ได้เท่ากับ -1, และ -2 จากนั้นคำนวณตามขั้นตอนการคำนวณที่ปรากฏตามรูปที่ 1 จะได้

$$\Lambda = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$$

$$V = \begin{bmatrix} -2 & -1 \\ 1 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} -1 & -1 \\ 1 & 2 \end{bmatrix}$$

$$W_{ij} = \int_0^\infty x_i x_j \, dt = \begin{bmatrix} 12.667 & -1 \\ -1 & 1.833 \end{bmatrix}$$

$$D = C^T C = \begin{bmatrix} 9 & 3 \\ 3 & 1 \end{bmatrix}$$

$$P = D*W = \begin{bmatrix} 111.003 & -3.501 \\ 37.001 & -1.117 \end{bmatrix}$$

เมื่อพิจารณาเฉพาะค่าแนวทะแยงมุมหลัก (Principle Diagonal) ของเมตริกซ์ P จะเห็นได้ว่าที่สถานะ x_1 ซึ่งสมนัยกับค่าไอเก็นเท่ากับ -1 จะแสดงผลทางพลังงานมากที่สุด ดังนั้นจะเลือกสถานะที่จะคงไว้ใน Reduced Model จากนั้นคำนวณค่าพารามิเตอร์ F,G,H และ L ของ Reduced Model ตามสมการที่ (3.67), (3.68), (3.71) และ (3.72) ตามลำดับ จะได้

$$F = -1$$

$$G = 6$$

$$H = 2.5$$

$$L = 2$$

เมื่อแทนค่า F และ G ลงในสมการที่ (3.30) ได้สมการสถานะเป็น

$$\dot{x} = -x + 6u \qquad ; x(0) = 0$$

จากนั้นแทนค่า H และ L ลงในสมการที่ (3.31) จะได้สมการเอาท์พุตเป็น

$$y = 2.5x + 2u$$

ถ้าให้อินพุตเป็น Unit Step Function จะได้

ระบบเดิม $\quad y(t) = 17 - 25e^{-t} + 8e^{-2t}$

Reduced Model $\quad y_r(t) = 17 - 15e^{-t}$

แต่ถ้าใช้วิธี Singular Perturbation ตามวิธีการเดิมซึ่งถ้าเราสามารถเลือกสถานะได้ถูกต้องคือเลือกสถานะได้เหมือนกับวิธีการที่เสนอในวิทยานิพนธ์ฉบับนี้ คือ เลือกสถานะ x_1 คงไว้โดยตัดสถานะ x_2 ออกไป จากนั้นคำนวณค่าพารามิเตอร์ F,G,H และ L โดยใช้วิธี Singular Perturbation โดยตรง จะต้องใช้สมการที่ (3.32) - (3.33) จะได้ค่าพารามิเตอร์เป็น

$$F = -0.667$$

$$G = 3.998$$

$$H = 2.667$$

$$L = 0.999$$

ได้สมการสถานะเป็น

$$\dot{x} = -0.667x + 3.998u$$

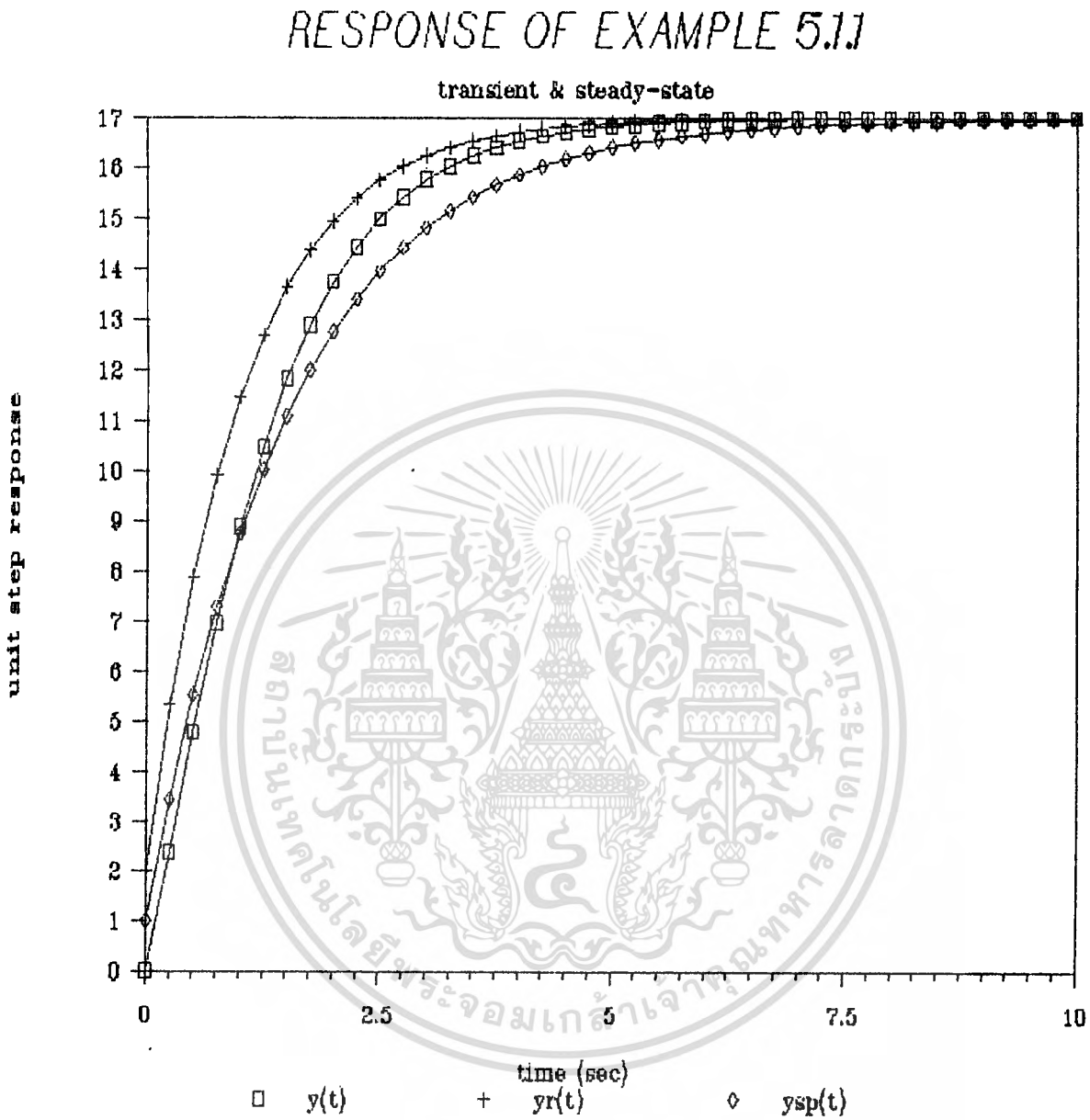
และสมการเอาท์พุตเป็น

$$y = 2.667x + 0.999u$$

ถ้าให้อินพุตเป็น unit step function จะได้เอาท์พุตเป็น

$$y_{sp}(t) = 16.992 - 15.997e^{-0.667t}$$

สำหรับผลตอบสนองเปรียบเทียบระบบในตัวอย่างนี้แสดงไว้ในรูปที่ 5 และรูปที่ 6 สำหรับค่าความผิดพลาดที่เกิดขึ้นแสดงไว้ในรูปที่ 7 และ รูปที่ 8 จะเห็นได้ว่าวิธีที่เสนอในวิทยานิพนธ์ให้ผลในช่วง steady - state ใกล้เคียงกับระบบเดิมมากกว่าวิธี SINGULAR

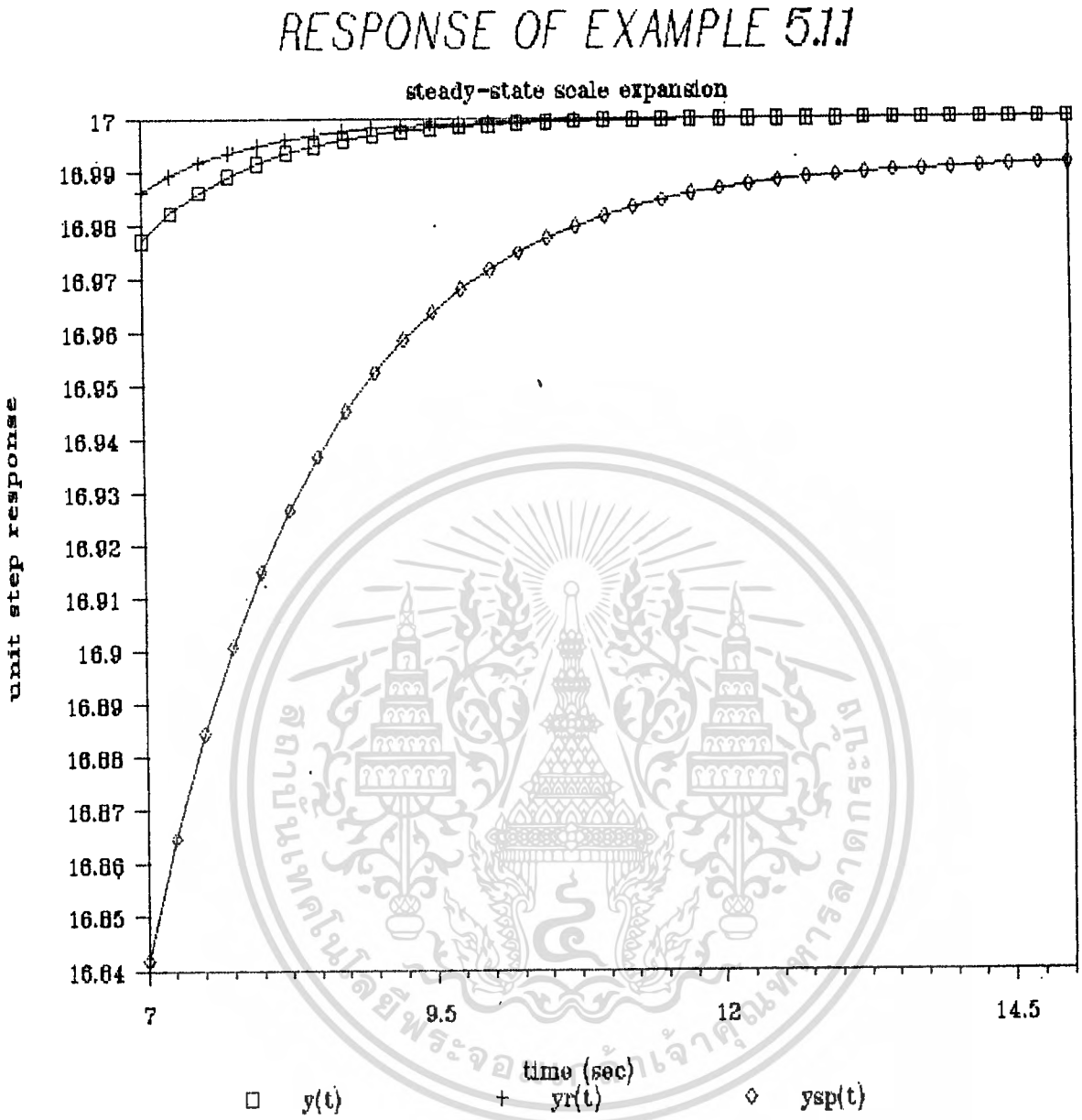


y(t) : ผลตอบสนองของระบบต้นแบบ

yr(t) : ผลตอบสนองของระบบ Reduced Model โดยวิธีที่เสนอในวิทยานิพนธ์

ysp(t) : ผลตอบสนองของระบบ Reduced Model โดยวิธี Singular Pertubation

รูปที่ 5 แสดงผลตอบสนองจากตัวอย่าง 5.1.1

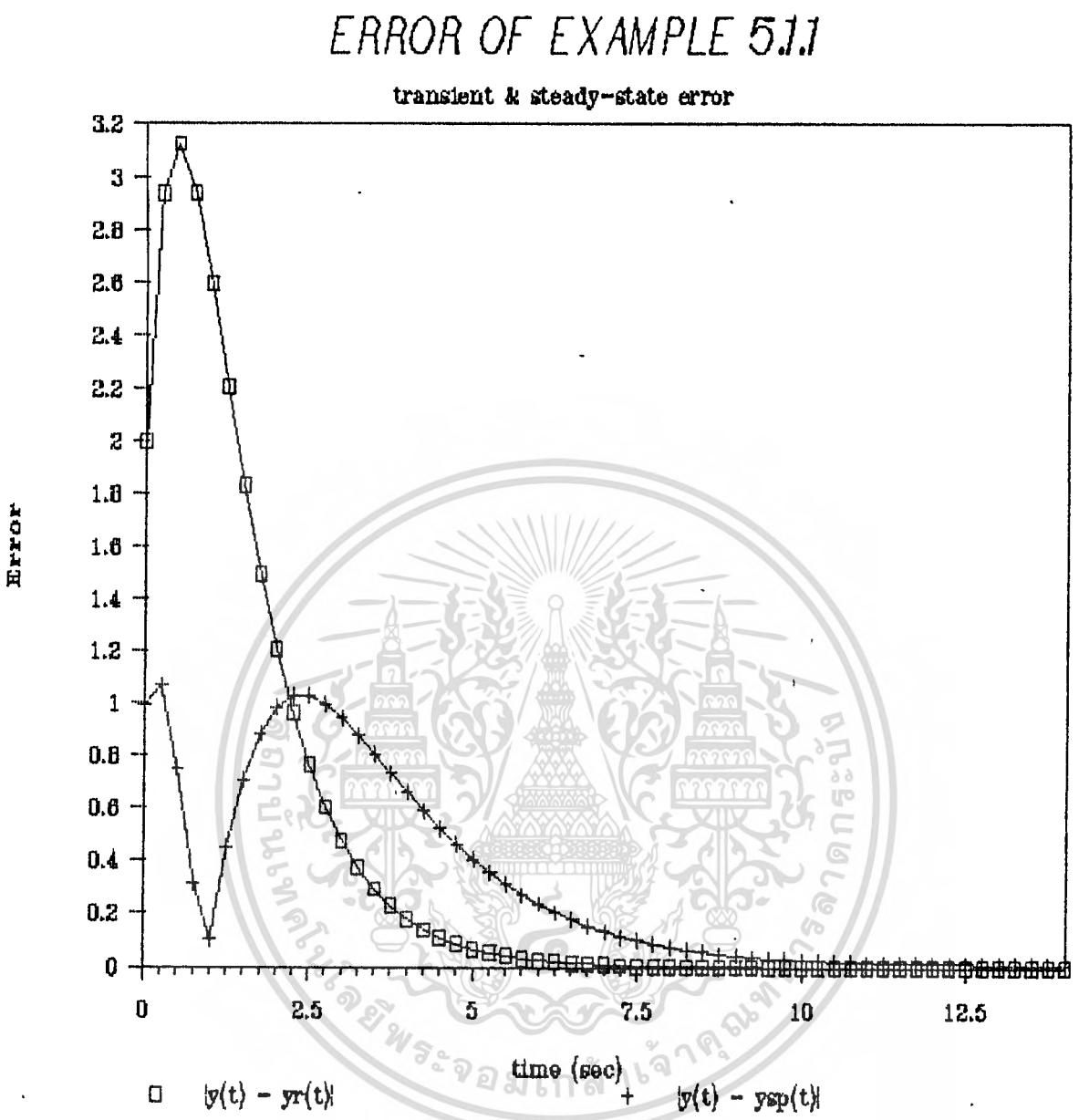


y(t) : ผลตอบสนองของระบบต้นแบบ

yr(t) : ผลตอบสนองของระบบ Reduced Model โดยวิธีที่เสนอในวิทยานิพนธ์

ysp(t) : ผลตอบสนองของระบบ Reduced Model โดยวิธี Singular Pertubation

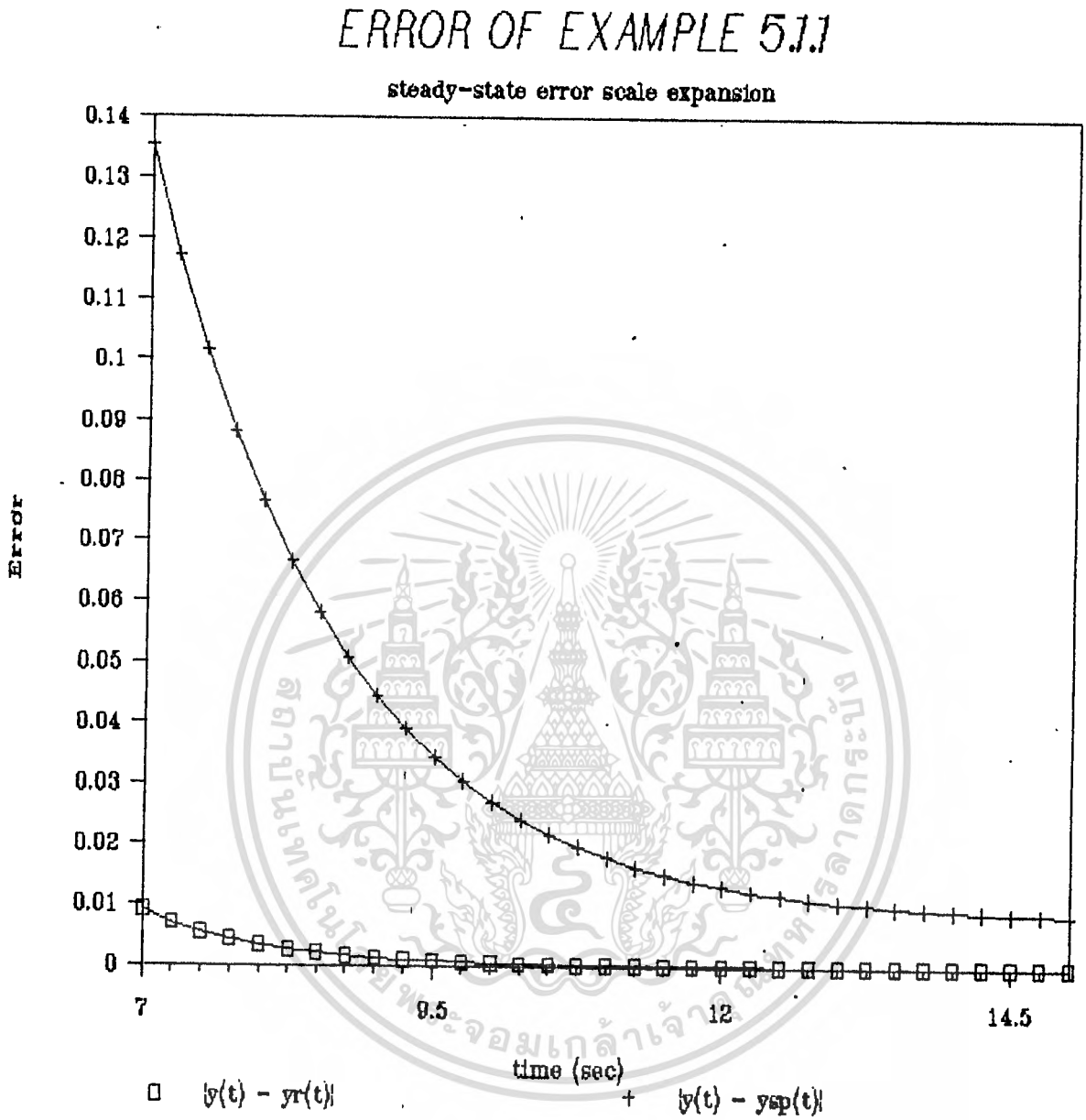
รูปที่ 6 แสดงผลตอบสนองในช่วง steady - state จากตัวอย่าง 5.1.1



|y(t) - yr(t)| : ค่าความผิดพลาดของระบบ Reduced Model โดยวิธีที่เสนอในวิทยานิพนธ์

|y(t) - ysp(t)| : ค่าความผิดพลาดของระบบ Reduced Model โดยวิธี Singular Perturbation

รูปที่ 7 ค่าความผิดพลาดของระบบ Reduced Model จากตัวอย่าง 5.1.1



|y(t) - yr(t)| : ค่าความผิดพลาดของระบบ Reduced Model
โดยวิธีที่เสนอในวิทยานิพนธ์

|y(t) - ysp(t)| : ค่าความผิดพลาดของระบบ Reduced Model
โดยวิธี Singular Pertubation

รูปที่ 8 ค่าความผิดพลาดของระบบ Reduced Model
ในช่วง steady - state จากตัวอย่าง 5.1.1

PERTURBATION มาก แต่ในช่วง transient จะผิดพลาดมากกว่า คาดว่าเกิดจากวิธีการเลือกสถานะที่เสนอจะตัดสถานะที่ส่งผลในช่วง transient ทิ้งไป

ตัวอย่าง 5.1.2

สมมุติให้เมตริกซ์ A, B และ C ในสมการที่ (3.1) และ (3.2) คือ

$$A=\begin{bmatrix} -0.91 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4.449 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10.262 & 571.749 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -571.749 & -10.262 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -10.487 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -15.214 & 11.622 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -11.622 & -15.214 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -89.874 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -502.665 \end{bmatrix}$$

$$B=\begin{bmatrix} -4.336 \\ -3.691 \\ 10.140 \\ -1.612 \\ 16.629 \\ -242.476 \\ -14.261 \\ 13.672 \\ 82.187 \end{bmatrix}$$

$$C=\begin{bmatrix} -0.422 & -0.736 & -0.004 & 0.232 & -0.816 & -0.715 & 0.546 & -0.235 & -0.081 \\ -0.003 & 0.0014 & 0.117 & -0.393 & 0.0011 & -0.045 & 0.069 & -0.08 & -0.129 \end{bmatrix}$$

เมื่อคำนวณค่า Participation Matrix ได้ค่าแนวทะแยงมุมหลักเป็น 20.6, 24.8, 0.0267, 0.0614, 94.4, 939, 239, 5.2, 2.08 เมื่อพิจารณาค่าเหล่านี้ ทำให้ได้สถานะที่ให้พลังงานมากและเป็นสถานะที่มีความสำคัญมากตามลำดับคือ x_6, x_7, x_5, x_2, x_1, x_8, x_9, x_4 และ x_3 ตามลำดับ

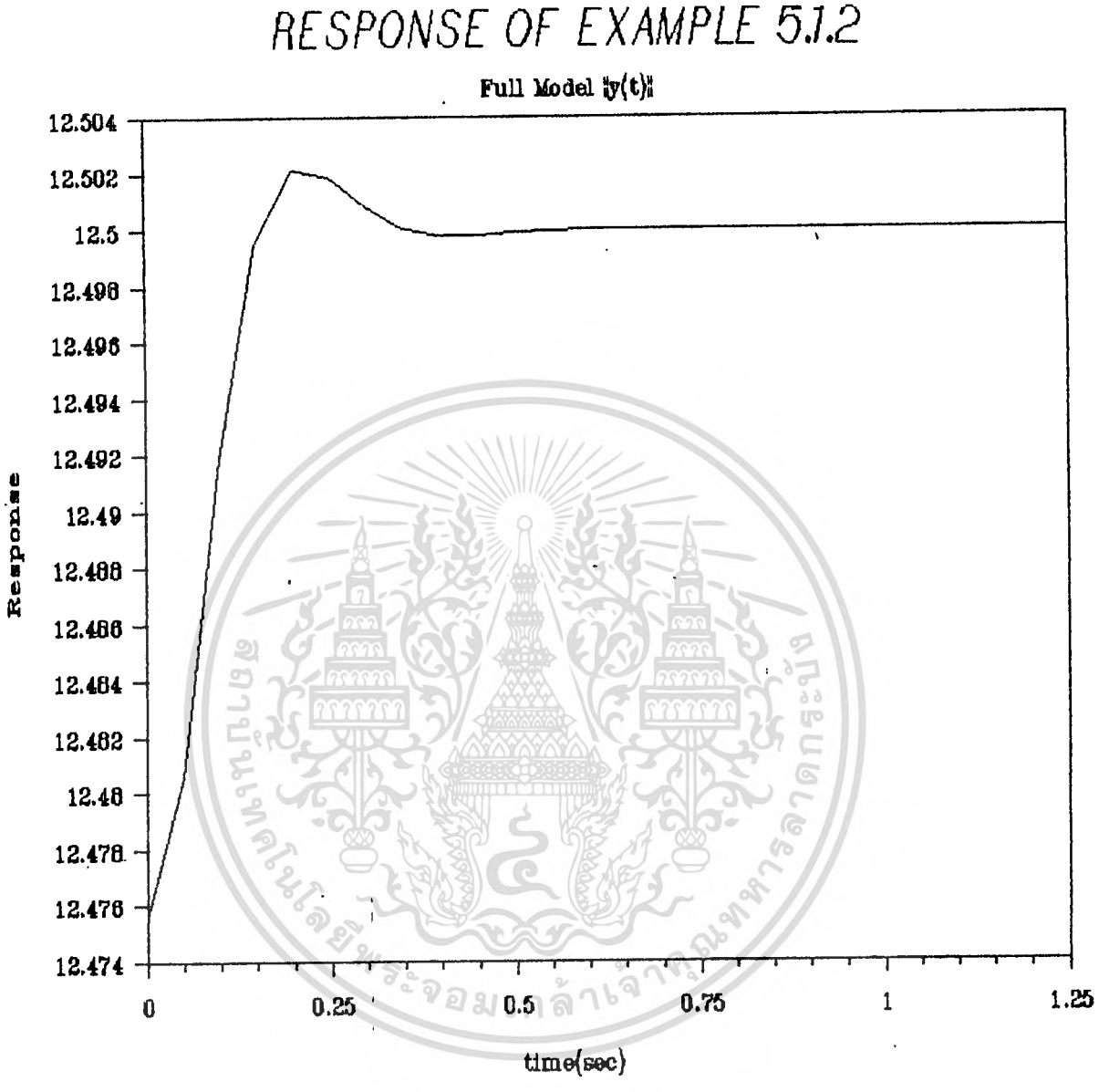
ถ้าเราเลือกสถานะให้มี Order เท่ากับ 2 จะได้สมการเสตทเป็น

$$\begin{bmatrix} \dot{x}_6 \\ \dot{x}_7 \end{bmatrix} = \begin{bmatrix} -15.214 & 11.622 \\ -11.622 & -15.214 \end{bmatrix} \begin{bmatrix} x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} -242.476 \\ -14.261 \end{bmatrix} u$$

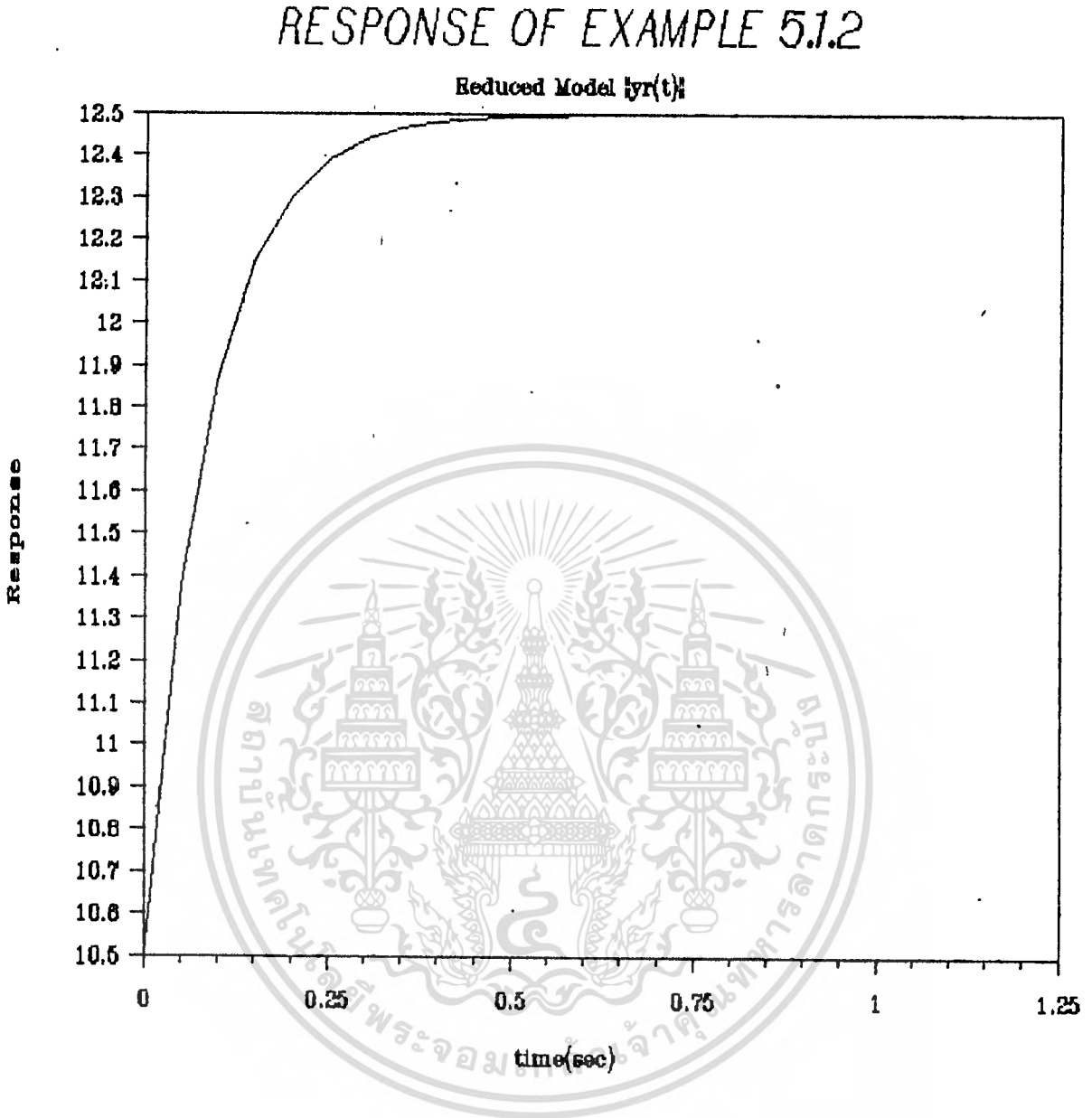
สำหรับสมการเอาท์พุตจะได้เป็น

$$y(t)= \begin{bmatrix} 0.715 & 0.546 \\ 0.045 & 0.0685 \end{bmatrix} \begin{bmatrix} x_6(t) \\ x_7(t) \end{bmatrix} + \begin{bmatrix} 1.33329313 \\ -0.0123269381 \end{bmatrix} u$$

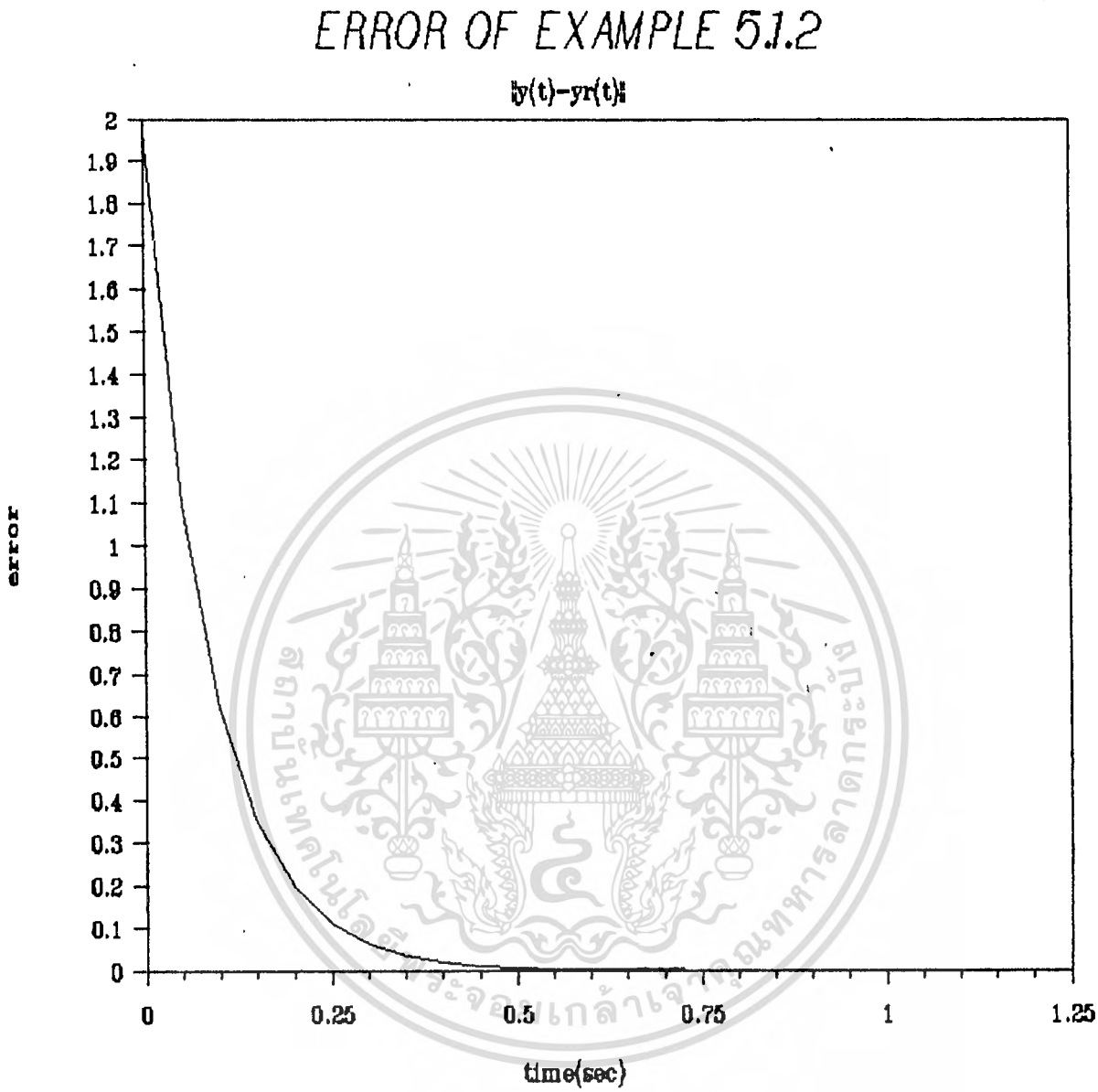
ผลตอบสนองของระบบต้นแบบในตัวอย่างนี้แสดงไว้ในรูปที่ 9 และเมื่อวิเคราะห์เป็น Reduced Model จะได้ผลตอบสนองตาม รูปที่ 10 ซึ่งแสดงค่า norm ของเอาท์พุตสำหรับค่าความผิดพลาดที่เกิดขึ้นแสดงไว้ในรูปที่ 11 จากรูปนี้แสดงให้เห็นว่า วิธีการที่เสนอในวิทยานิพนธ์นี้มีค่าความผิดพลาดมากในช่วง transient โดยการพิจารณาจะเห็นว่า ในช่วงนี้ ผลตอบสนองของระบบต้นแบบ มี overshoot เกิดขึ้น แต่ผลตอบสนองของระบบ MODEL REDUCTION ส่วน overshoot จะหายไป สำหรับช่วง steady - state ระบบ MODEL REDUCTION จะให้ผลตอบสนองที่ใกล้เคียงกับระบบเดิมมาก



รูปที่ 9 แสดงผลตอบสนองของระบบต้นแบบจากตัวอย่าง 5.1.2



รูปที่ 10 แสดงผลตอบสนองของ Reduced Model จากตัวอย่าง 5.1.2



รูปที่ 11 แสดงค่าความผิดพลาด จากตัวอย่าง 5.1.2

5.2 การทดสอบระบบในโดเมนของความถี่

ในบทที่ 4 ที่ผ่านมาจะเห็นได้ว่า การสร้าง Model Reduction ในโดเมนของความถี่ก็คือการลดทอน Order ของ transfer function ของระบบนั่นเอง ปัญหาในโดเมนของความถี่นั้นคือ การหาค่าสัมประสิทธิ์ของ transfer function หลังจากการเลือกจำนวนสถานะไว้แล้ว และระบบที่ได้นี้จะต้องเสถียรด้วย ถ้าระบบเดิมเสถียร ในการแก้ปัญหานี้ในบทที่ 4 ได้เสนอวิธีการประมาณค่าแบบ Pade´ มาประยุกต์ใช้ร่วมกับวิธี Fitting Time - Moments

สำหรับในหัวข้อนี้ จะแสดงตัวอย่างบางส่วนที่ได้นำมาทดสอบระบบของ Reduced Model ตามวิธีการที่ได้เสนอไว้แล้วในบทที่ 4

ตัวอย่าง 5.2.1

พิจารณาระบบที่มีสมการ transfer function มี Order เท่ากับ 3 และมีรูปแบบเป็น

$$G(s) = \frac{8s^2 + 6s + 2}{s^3 + 4s^2 + 5s + 2}$$

ระบบนี้มีโพลอยู่ที่ -1,-1 และ -2 เมื่อเรากระจาย G(s) ให้อยู่ในรูปของอนุกรมกำลัง

$$G(s) = c_o + c_1 s + c_2 s^2 + \ldots$$

จากนั้นใช้การประมาณค่าแบบ Pade´ จะได้ค่า c_1 เป็น

$$c_o = 1$$

$$c_1 = 0,5$$

$$c_2 = 0.75$$

ถ้าต้องการให้ Order ของ Reduced Model ให้มีค่าเท่ากับ 2 และเลือกค่าโพลที่จะคงไว้เท่ากับ -1 จะพบว่าระบบ Reduced Model ที่ได้ไม่เสถียร แต่ถ้าเลือกค่าโพลที่จะคงไว้เท่ากับ -2 จะได้ระบบ Reduced Model ที่เสถียร จากนั้นคำนวณค่าสัมประสิทธิ์ของ Reduced Model ตามวิธี Fitting Time - Moment จะได้

$$a_o = 6$$

$$a_1 = 8$$

$$b_o = 6$$

$$b_1 = 5$$

ดังนั้นจะได้สมการ transfer function ของ Reduced Model ที่ใช้แทนระบบเดิมเป็น

$$[R_{fm}(s)]_2 = \frac{8s + 6}{s^2 + 5s + 6}$$

แต่ถ้าคำนวณตามวิธี Stability - Equation โดยให้ $\alpha = 2$ จะได้สมการ transfer function เป็น

$$[R_{se}(s)]_2 = \frac{1.5s + 0.5}{s^2 + 1.25s + 0.5}$$

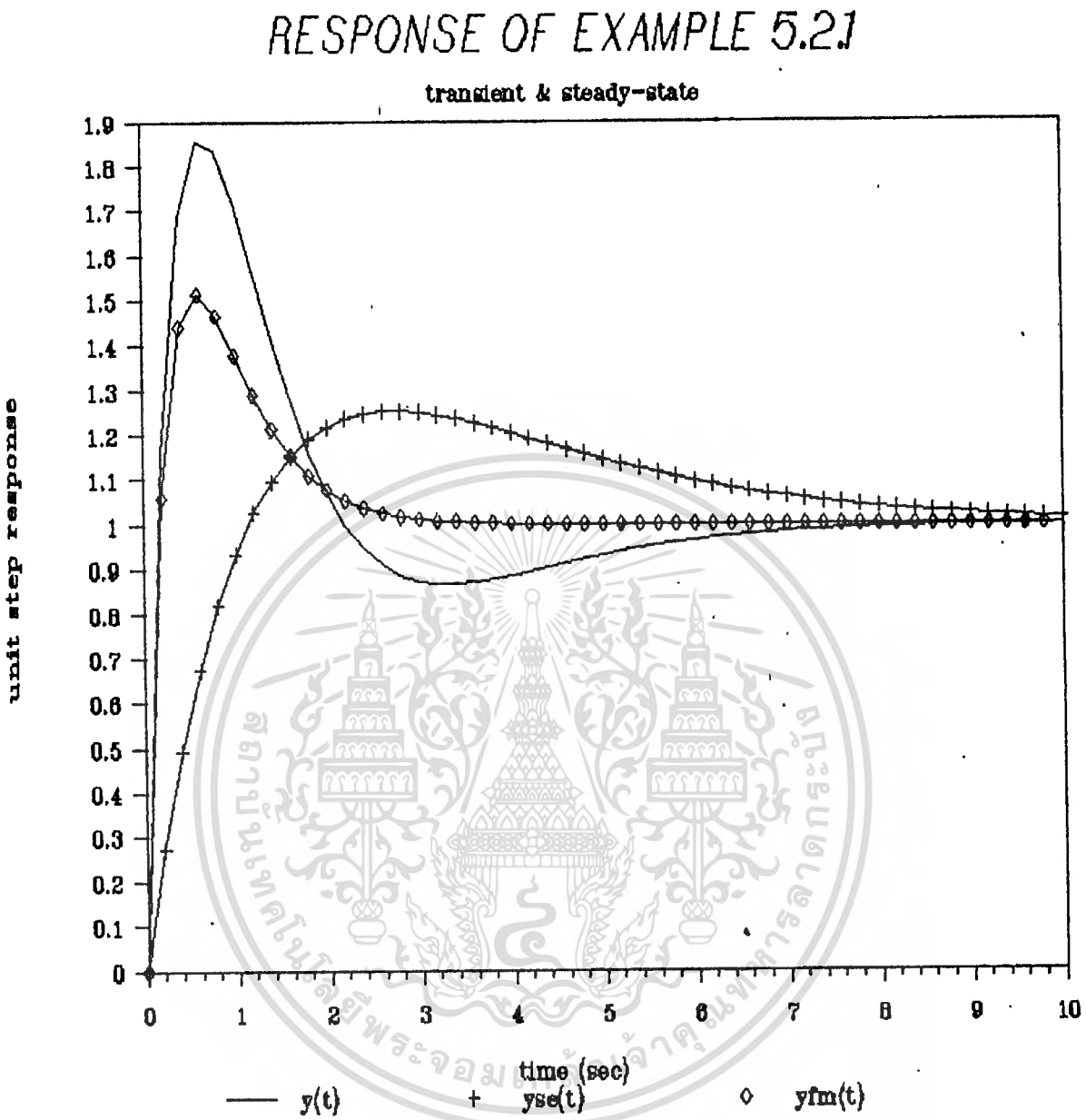
สำหรับตัวอย่างนี้ ถ้าให้อินพุต เป็น unit step function จะได้ผลตอบสนองของระบบต้นแบบ $\{y(t)\}$, ระบบ Reduced Model โดยวิธี Fitting Time - Moments $\{y_{fm}(t)\}$ และวิธี Stability - Equation $\{y_{se}(t)\}$ คือ

$$y(t) = 0.5 + (5-2t)e^{-t} - 5.5e^{-2t}$$

$$y_{fm}(t) = 1 - 6e^{-3t} + 5e^{-2t}$$

$$y_{se}(t) = 1-2e^{-0.625t}\{0.5\cos(0.331t) - 1.132\sin(0.331t)\}$$

สำหรับผลตอบสนองของระบบต้นแบบ G(s) กับระบบ Reduced Model แสดงไว้ในรูปที่ 12 สำหรับค่าความผิดพลาดของทั้ง 2 วิธี แสดงไว้ในรูปที่ 13 จะเห็นว่า Reduced Model ทั้งวิธี Fitting Time-Moments และวิธี Stability-Equation ให้ผลตอบสนองใกล้เคียงกับระบบต้นแบบ แต่วิธี Fitting Time - Moments สามารถนำไปพัฒนาเป็นโปรแกรมได้สะดวกกว่าวิธี Stability - Equation มาก

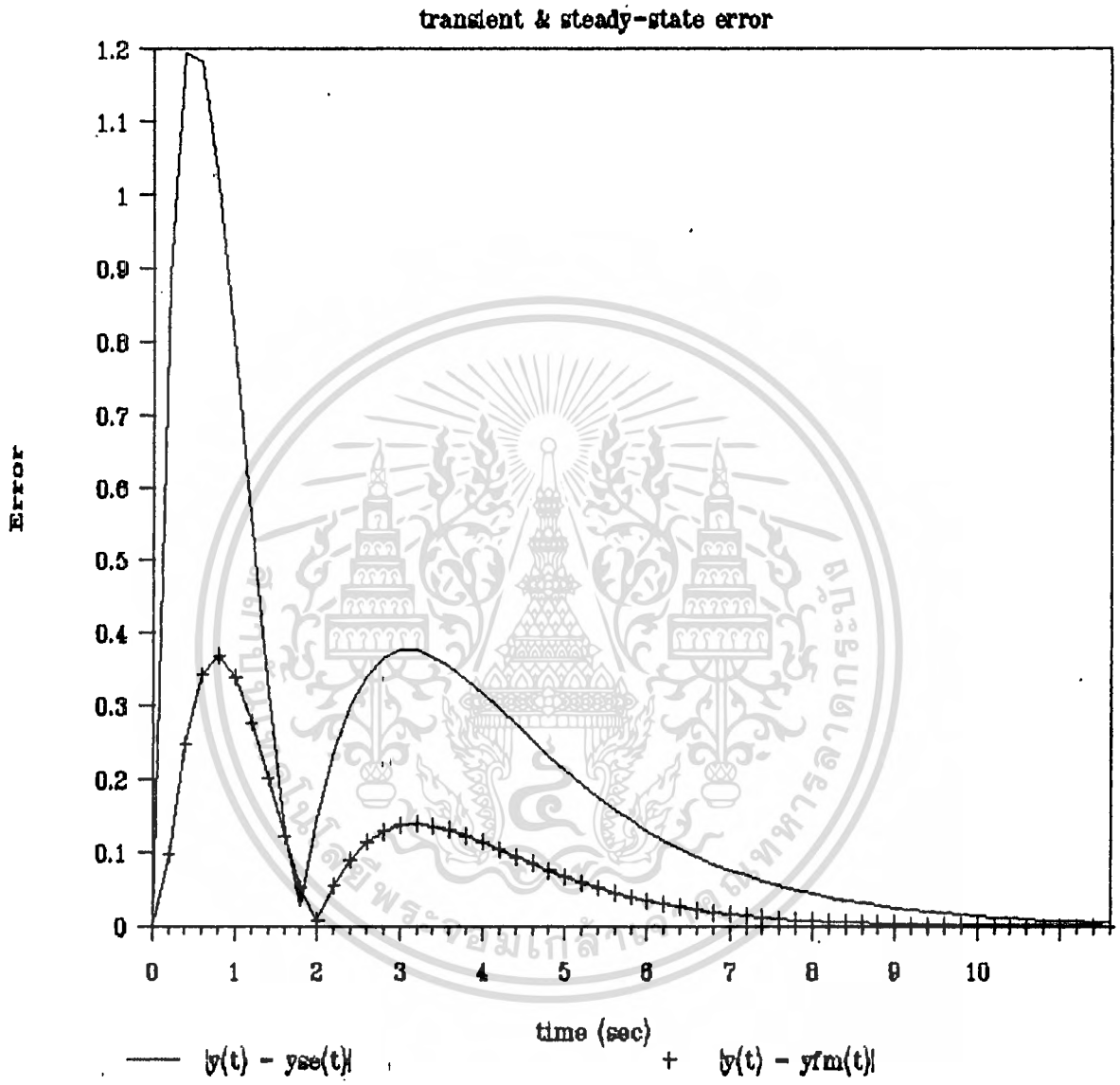


y(t) :ผลตอบสนองของระบบต้นแบบ

yse(t):ผลตอบสนองของระบบ Reduced Model โดยวิธี Stability-Equation

yfm(t):ผลตอบสนองของระบบ Reduced Model โดยวิธี Fitting Time-Moments

รูปที่ 12 แสดงผลตอบสนองจากตัวอย่าง 5.2.1



|y(t)-yse(t)| : ค่าความผิดพลาดของ Reduced Model จาก วิธี Stability - Equation

|y(t)-yft(t)| : ค่าความผิดพลาดของ Reduced Model จาก วิธี Fitting Time-Moments

รูปที่ 13 แสดงค่าความผิดพลาดจากตัวอย่าง 5.2.1

ตัวอย่าง 5.2.2

สำหรับตัวอย่างนี้จะแสดงถึงผลของการลดทอน Order ของ transfer function ว่า ถ้า Reduced Model ที่ได้มี Order ต่ำลงมาก ๆ จะมีผลตอบสนองแตกต่างจากระบบต้นแบบอย่างไร

พิจารณาระบบที่มีสมการ transfer function มี Order เท่ากับ 6 และมีรูปแบบเป็น

$$G(s) = \frac{s^5+1014s^4+1469s^3+6914s^2+140100s+100000}{s^6+2225s^5+14541s^4+248420s^3+1454100s^2+2220000s+1000000}$$

ระบบนี้มีค่าโพลเท่ากับ -1, -1, -10, -10, -100, -100 เมื่อลดทอน Order ของระบบโดยใช้วิธีการประมาณค่าแบบ Pade' ให้เหลือ Order เท่ากับ 2 หรือ 3 จะพบว่าระบบที่ได้ไม่เสถียร

เมื่อใช้วิธี Fitting Time - Moments โดยใช้วิธีการประมาณค่าแบบ Pade' คำนวณค่า c_i ได้เป็น

$$c_0 = 0.1$$
$$c_1 = -0.0819$$
$$c_2 = 0.0433$$
$$c_3 = -0.0005$$
$$c_4 = -0.0421$$
$$c_5 = -0.0843$$

ถ้าเราต้องการลดทอน Order ให้เหลือ Order เท่ากับ 2 และเลือกโพลเท่ากับ -1 เพื่อจะคงไว้ จะพบว่าระบบ Reduced Model ที่ได้ไม่เสถียร แต่ถ้าเลือกโพลที่จะคงไว้เป็น -100 จะได้ระบบที่เสถียร และสมการ transfer function ของ Reduced Model เป็น

$$[R(s)]_2 = \frac{3.755072s + 7.7247}{s^2 + 100.77247s + 77.247}$$

สำหรับการสร้างสมการ transfer function ของ Reduced Model ที่มี Order เท่ากับ 3 จะพบว่า เราต้องใช้ค่าโพลที่ -100 จึงจะได้ระบบที่เสถียร และสมการ transfer function ที่ได้มีรูปแบบเป็น

$$[R(s)]_3 = \frac{5.3673s^2 + 15.28212s + 19.68001}{s^3 + 103.12032s^2 + 314.0005s + 196.8001}$$

สำหรับตัวอย่างนี้ ถ้าให้อินพุต เป็น unit step function จะได้ผลตอบสนองของระบบต้นแบบ $\{y(t)\}$, ระบบ Reduced Model ที่มี Order เท่ากับ 2 และ 3 คือ

ระบบต้นแบบ

$$y(t) = 0.1 + (-0.126 + 0.041t)e^{-t} + (0.179 - 1.298t)e^{-10t} + (-0.153 - 11.405t)e^{-100t}$$

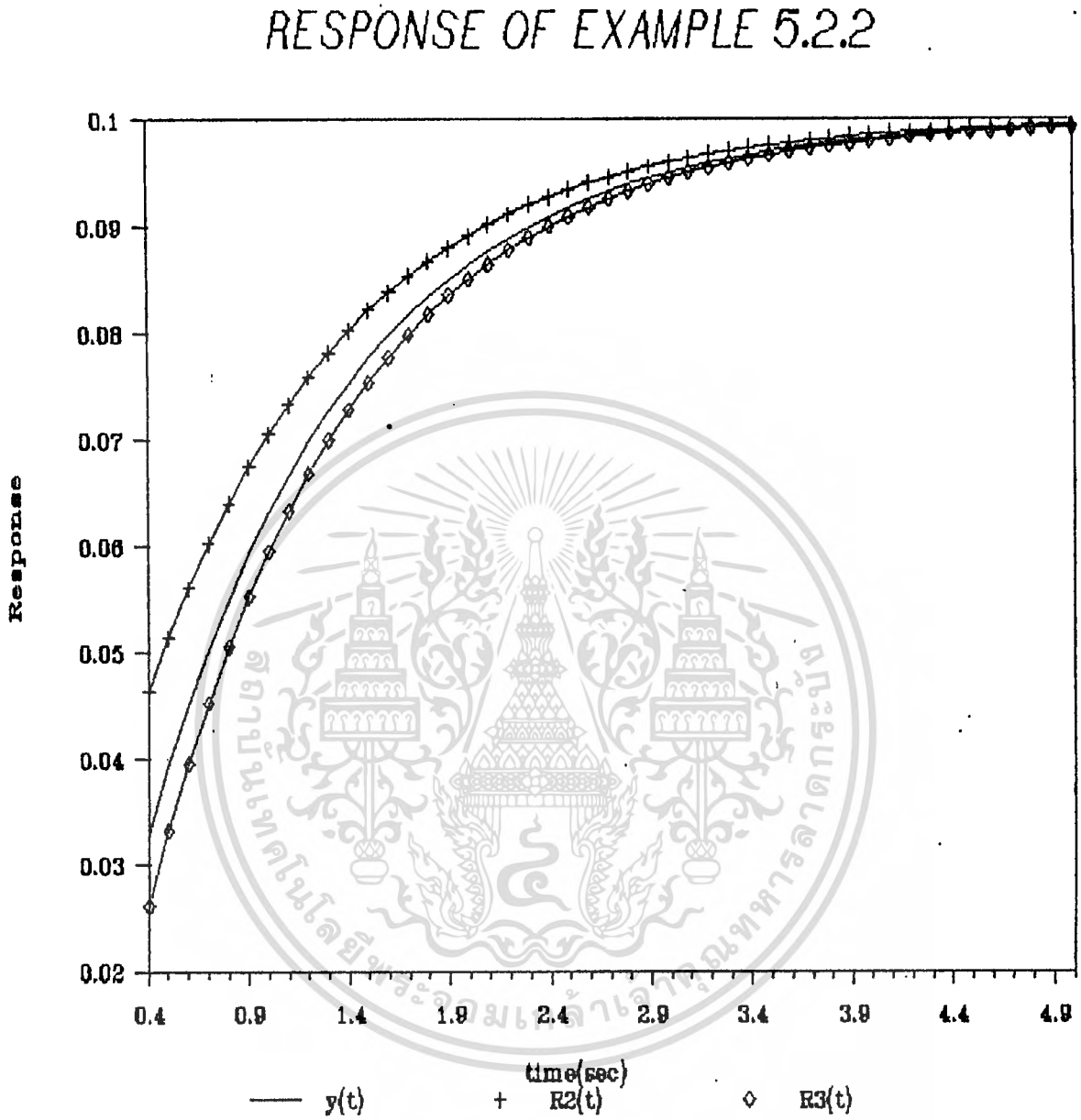
Reduced Model ที่มี Order เท่ากับ 2

$$R_2(t) = 0.1 - 0.063e^{-0.77t} - 0.037e^{-100}$$

Reduced Model ที่มี Order เท่ากับ 3

$$R_3(t) = 0.1-0.088e^{-0.877t}+0.41e^{-2.243t}-0.054e^{-100t}$$

ผลตอบสนองเปรียบเทียบระหว่าง Reduced Model ที่มี Order เท่ากับ 2 และ 3 กับระบบต้นแบบ แสดงไว้ในรูปที่ 14 ค่าความผิดพลาดของ Reduced Model ที่ Order 2 และ Order 3 แสดงไว้ในรูปที่ 15 เมื่อพิจารณาจะเห็นว่าค่าความผิดพลาดของ Reduced Model จะมีค่ามากขึ้นตามจำนวน Order ที่ลดทอน

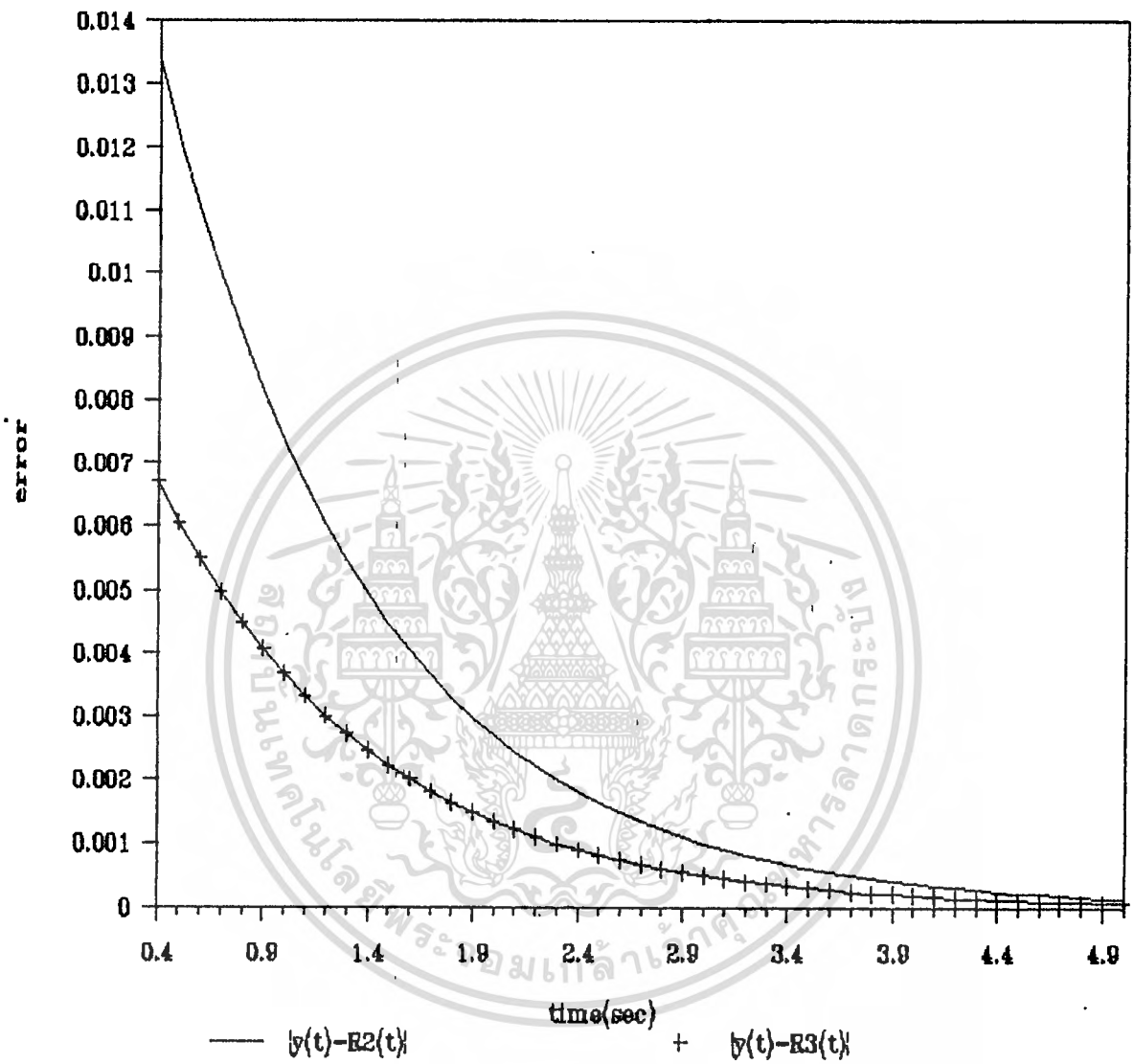


y(t) :ผลตอบสนองของระบบต้นแบบ

R2(t):ผลตอบสนองของระบบ Reduced Model ที่มี Order เท่ากับ 2

R3(t):ผลตอบสนองของระบบ Reduced Model ที่มี Order เท่ากับ 3

รูปที่ 14 แสดงผลตอบสนองจากตัวอย่าง 5.2.2



|y(t)-R2(t)| : ค่าความผิดพลาดของ Reduced Model ที่มี Order เท่ากับ 2

|y(t)-R3(t)| : ค่าความผิดพลาดของ Reduced Model ที่มี Order เท่ากับ 3

รูปที่ 15 แสดงค่าความผิดพลาดจากตัวอย่าง 5.2.2

บทที่ 6

บทสรุปและข้อเสนอแนะ

เมื่อทำการทดสอบระบบของ Reduced Model ตามวิธีที่ได้ทำการศึกษาในวิทยานิพนธ์ฉบับนี้ โดยในโดเมนของเวลา จะใช้วิธีการพิจารณาพลังงานอิมพัลซ์ในการเลือกสถานะที่จะคงไว้ จากนั้นจะนำวิธี Singular Perturbation มาประยุกต์ใช้ในการหาค่าพารามิเตอร์ของระบบ ส่วนในโดเมนของความถี่ ใช้วิธี Fitting Time - Moments ร่วมกับการประมาณค่าแบบ Pade´ จะเห็นได้ว่า สามารถแยกการวิเคราะห์ออกเป็นสองส่วนคือ

1. ช่วง Transient State เป็นช่วงเริ่มต้นที่ระบบยังไม่มีค่าเข้าสู่ค่าใดค่าหนึ่ง ในช่วงนี้ เมื่อพิจารณาจากผลตอบสนองและกราฟแสดงค่าความผิดพลาด ทั้งในโดเมนของเวลาและในโดเมนของความถี่ จะเห็นได้ว่า วิธีต่าง ๆ ที่ทำการศึกษาในวิทยานิพนธ์ฉบับนี้ มีความผิดพลาดค่อนข้างมาก ซึ่งคาดว่าอาจเกิดจากการตัดสถานะบางสถานะทิ้งไป โดยที่สถานะเหล่านี้อาจเป็นสถานะที่ส่งผลในช่วง transient ดังนั้น เมื่อทำการสร้างตัวแบบเป็น Reduced Model และนำผลตอบสนองที่ได้ไปเปรียบเทียบกับระบบเดิม จึงมีความผิดพลาดในช่วงนี้ค่อนข้างสูง

2. ช่วง Steady State เป็นช่วงที่ระบบมีค่าเข้าใกล้ค่าคงที่ค่าหนึ่ง สำหรับผลตอบสนองของระบบ Reduced Model เมื่อเปรียบเทียบกับระบบเดิมในโดเมนของเวลา เราจะพบว่า ในวิธี Singular Perturbation ถ้าเลือกสถานะที่จะคงไว้เป็นสถานะเดียวกับวิธีที่เสนอในวิทยานิพนธ์ฉบับนี้ ในช่วงนี้จะมีความผิดพลาดสูงกว่าวิธีที่เสนอ คาดว่าเกิดจากไม่ได้นำเอาเมตริกซ์ B และ C มาช่วยในการวิเคราะห์และในวิธี Singular Perturbation นี้จะมีความผิดพลาดในช่วง Steady State มากขึ้นอีก ถ้าเลือกสถานะไม่เหมาะสม สำหรับในโดเมนของความถี่ เมื่อทำการทดสอบระบบหลาย ๆ ระบบจะพบว่าค่าความผิดพลาดในช่วงนี้ ทั้งวิธี Stability - Equation และวิธี Fitting Time - Moments จะมีค่าความผิดพลาดใกล้เคียงกัน ผลตอบสนองทั้ง 2 วิธีนี้จะมีค่าใกล้เคียงกับระบบเดิมมาก อย่างไรก็ตาม เมื่อพิจารณาจากขั้นตอนการคำนวณที่แสดงไว้ในบทที่ 4 จะเห็นได้ว่า เมื่อประยุกต์วิธีการประมาณค่าแบบ Pade´ ในการ

คำนวณค่า time - moments แล้ว จะทำให้การพัฒนาโปรแกรมสะดวกกว่าวิธี Stability - Equation มาก และเวลาในการประมวลผลจะเร็วกว่าเพราะในวิธี Fitting Time - Moments ไม่ต้องแยกคำนวณค่าโพลที่เป็นจำนวนคู่หรือจำนวนคี่ และการคำนวณค่า time - moments กับค่าสัมประสิทธิ์ของ Reduced Model จะใช้การแก้สมการเชิงเส้นเท่านั้น นอกจากนั้นยังให้ผลตอบสนองใกล้เคียงกับวิธี Stability - Equation ซึ่งมีค่าใกล้เคียงกับระบบเดิม แต่ระบบที่นำมาทดสอบทั้งในโดเมนของเวลา และในโดเมนของความถี่ ตัวแบบของ Reduced Model ที่ได้ ส่วนมากจะมีค่าความผิดพลาดมากขึ้นตามจำนวนตัวแปรสถานะ หรือจำนวน Order ที่ทำการลดทอน

จากการวิเคราะห์ที่ผ่านมาเราพบว่าการพัฒนาโปรแกรม Reduced Model ในโดเมนของความถี่ สะดวกกว่าในโดเมนของเวลามาก และสามารถเปรียบเทียบตัวแบบที่ได้จาก Reduced Model และตัวแบบที่ได้จากระบบเดิมตามตารางที่ 6.1 คือ

ตัวแบบของระบบเดิม	ตัวแบบของ Reduced Model
1. ในโดเมนของเวลา จำนวนของตัวแปรสถานะมีจำนวนมาก	1. ในโดเมนของเวลา สามารถลดจำนวนของตัวแปรสถานะให้เหลือตามความต้องการ ทำให้จำนวนของตัวแปรสถานะมีจำนวนน้อย
2. ในโดเมนของความถี่จะมี Order ของสมการ transfer function สูง	2. ในโดเมนของความถี่ สามารถลด Order ของสมการ transfer function ให้เหลือได้ตามต้องการ

ตารางที่ 6.1 แสดงการเปรียบเทียบข้อดี และข้อเสีย ของตัวแบบที่ได้จากระบบเดิมและระบบ Reduced Model

ตัวแบบของระบบเดิม	ตัวแบบของ Reduced Model
3. การวิเคราะห์ระบบกระทำได้ไม่สะดวก เพราะมีตัวแปรสถานะ ในโดเมนของเวลา และจำนวน Order ของสมการ transfer function ในโดเมนของความถี่ค่อนข้างมาก	3. เนื่องจากมีตัวแปรสถานะในโดเมนของเวลา และมี Order ของสมการ transfer function ในโดเมนของความถี่มีจำนวนน้อยทำให้การวิเคราะห์ระบบกระทำได้ง่าย
4. เมื่อทำการสังเคราะห์ระบบ อาจใช้อุปกรณ์ต่าง ๆ เป็นจำนวนมาก	4. ช่วยลดจำนวนอุปกรณ์ เมื่อต้องการสังเคราะห์ระบบ
5. เมื่อทำการวิเคราะห์ระบบ จะได้ผลของการวิเคราะห์ที่มีความถูกต้องสูง	5. เมื่อทำการวิเคราะห์ระบบ อาจมีความผิดพลาดเกิดขึ้นได้ โดยเฉพาะในช่วง transient state จะมีความผิดพลาดค่อนข้างมาก แต่ในช่วง steady state ค่าความผิดพลาดจะเกิดขึ้นเล็กน้อย และจะเพิ่มมากขึ้นตามจำนวนตัวแปรสถานะหรือ Order ที่ลด

ตารางที่ 6.1 (ต่อ) แสดงการเปรียบเทียบข้อดี และข้อเสีย ของตัวแบบที่ได้จากระบบเดิม และระบบ Reduced Model

ข้อเสนอแนะที่จะศึกษาและพัฒนาต่อไป

ข้อเสนอแนะสำหรับผู้ที่สนใจจะทำวิจัยต่อจากวิทยานิพนธ์ฉบับนี้ คือ

1. ศึกษาและพัฒนาโปรแกรม ที่ช่วยในการวิเคราะห์ การลดทอนตัวแบบในระบบที่ไม่ต่อเนื่องกับเวลา (Discrete - Time Systems)
2. ศึกษาและทดลองถึงการนำเอาวิธี Reduced Model ไปทดลองสังเคราะห์ระบบควบคุม เปรียบเทียบกับระบบเดิม

กิตติกรรมประกาศ

ขอขอบพระคุณ รองศาสตราจารย์ วินัน์ ปรีชาพานิช เป็นอย่างสูงที่ได้ประสิทธิ์ประสาทวิชาการให้แก่ผู้เขียนตลอดเวลาที่ได้ทำการศึกษาอยู่ที่สถาบันแห่งนี้ อีกทั้งให้คำแนะนำปรึกษาในการแก้ปัญหาต่าง ๆ จนกระทั่งวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี ขอขอบพระคุณ ศาสตราจารย์ ดร. ไพรัช ธัชยพงษ์ และเจ้าหน้าที่สำนักวิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง ทุกท่านที่เอื้อเฟื้ออุปกรณ์ที่ใช้ในการทำวิทยานิพนธ์ ขอขอบคุณ นักศึกษาปริญญาโท ภาควิชาวิศวกรรมระบบควบคุม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง ทุกคนที่ให้การสนับสนุนผู้เขียนด้วยดีเสมอมา

วิทยานิพนธ์ฉบับนี้ได้รับการสนับสนุนเงินทุนในการศึกษาและวิจัยจาก มูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร (C & C EDUCATION FOUNDATION) จนกระทั่งเป็นผลสำเร็จ ผู้รับทุนขอขอบพระคุณมา ณ โอกาสนี้ด้วย

เอกสารอ้างอิง

1. Antonio Lepschy, Gian A. Mian, and Umberto Viaro, "Stability Preservation and Computational Aspecta of a Newly Proposed Reduction Method", IEEE Trans Automat. Contr., Vol.33, No.3, Mar.1988, pp.307-309

2. Aoki, M., "Control of Large-Scale Dynamic Systems by Aggregation", IEEE Trans Automat. Contr., Vol. AC-13, No.3, Jun. 1968, pp.246-253

3. Aoki, M., "Some Approximation Methods for Estimation and Control of Large Scale Systems", IEEE Trans Automat. Contr., Vol. AC-23, No.2, Apr. 1978, pp.173-182

4. Bai-Wu Wan, "Linear model reduction using Mihailov criterion and Pade approximation technique", Int. J. Control, Vol. 33, NO.6, 1981, pp.1073-1089

5. Bistritz Y. and Langholz G., "Model Reduction by Chebyshev Polynomial Techiques", IEEE Trans Automat. Contr., Vol.AC-24, No.5 Oct.1979, pp.741-747

6. Bonvin, D., and Mellichamp, D.A., "A unified derivation and critical review of modal approaches to model reduction", Int. J. Cont, 1982, Vol.35, No.5, pp. 829-848

7. Chen, T.C. and Chan, C.Y., "Stable reduced-order Pade approximants using stability-equation method", Electron. Lett., Vol.16, No.19, Apr. 1980, pp-345-346

8. Chyi Hwang and Muh-Yang Chen, "A Multipoint Continued-Fraction Expansion for Linear System Reduction", IEEE Trans Automat. Contr., Vol. AC-31, No.7, Jul 1986, pp.648-651

9. Daniels, R.W., "Approximation Methods for Electronic Filter Design", McGraw-Hill, New York, 1974

10. Davison, E.J., "A method for simlifting linear dynamic systems", IEEE Trans Automat. Contr., 1966, AC-11, pp.93-101

11. Davison, E.J., "A New Method for Simlifting Large Linear Dynamic Systems", IEEE Trans Automat. Contr., Apr. 1968, AC-13, pp.214-215

12. Gerhard Kreisselmeier and Manfred Mevenkamp, "A Note Reduced-Order Controller Synthesis", IEEE Trans Automat. Contr., Vol.33, No.9, Sep. 1988, pp.878-880

13. Grodt, T. and Gajic, Z., "The Recursive Reduced-Order Numerical Solution of the Singularly Perturbed Matrix Differential Riccati", IEEE Trans Automat. Contr., Vol.33, No.8, Aug. 1988, pp. 751-754

14. Helmuth Stahl and Peter Hippe, "Comment on FF-Pade Method of Model Reduction in Frequency Domain", IEEE Trans Automat. Contr., Vol.33, No.4, Apr. 1988, pp.415-416

15. Hickin, J. and Sinha N.K., " Model Reduction for Linear Multivariable Systems", IEEE Trans Automat. Contr., Vol. AC-25, No.6, Dec. 1980

16. Hu Xiheng, "FF - Pade Method of Model Reduction in Frequency Domain", IEEE Trans Automat. Contr., Vol. AC-32, No. 3, Mar. 1987, pp.243-246

17. Hutton, M. F. and Friedland, GB., "Routh Approximation for Reduction Order of Linear, Time-Invariant Systems", IEEE Trans. Autom. Contral., AC-20 1975, pp.329-337

18. Jamshidi, M. and Malek-Zavarei, M., "LINEAR CONTROL SYSTEMS A Computer-Aided Approach", Pergamon Press, Great Britain, 1986

19. Jium-Ming Lin and Kuang-Wei Han, "Reduction the Effect of Model Reduction on Stability Boundaries and Limit-Cycle Characteristic", IEEE Trans. Automat. Contr., Vol. AC-31, No.6 Jun. 1986, pp.567-569

20. Kung, S.Y. and Lin, D.W., "Optimal Hankel-Norm Model Reductions; Multivariable Systems", IEEE Trans. Automat. Contr., Vol. AC-26, No.6 1981, pp.832-852

21. Lal, M. and Mitra, R., "Simplication of Large System Dynamic Using a Moment Evalution Algorithm", IEEE Trans. Automat. Contr., Vol. AC-9 Oct. 1974, pp.602-603

22. Lamba, S.S. and Vittal Rao, S. "On Suboptimal Control via the Simplified Model of Davison", IEEE Trans Automat. Contr., Aug. 1974, pp. 448-450

23. Lastman, G.J., and Sinha, N.K., "On the selection of states to be retained in d reduced-order model", Proc. IEE, Jan. 1984, Vol 131 pp.15 - 22

24. Lucas, T.N., "EFFICIENT ALGORITHM FOR REDUCTION BY CONTINUED - FRACTION EXPANTION ABOUT s = 0 AND s = a" Electron. Lett., Vol.19, No.23, Nov. 1983, pp.991-993

25. Lucas, T.N., "LINEAR SYSTEM REDUCTION BY CONTINUED - FRACTION EXPANTION ABOUT s = 0 AND s = a ALTERNATELY", Electron. Lett., Vol.19, No.7, Feb. 1983, pp.244-246

26. Lucas, T.N., "Linear System Reduction by Impulse Energy Approximation", IEEE Trans Automat. Contr., Vol. AC-30, No.8, Aug. 1985, pp.784-786

27. Ouyang, M., Liaw, C.M. and Pan C.T., "Model Reduction by Power Decomposition and Frequency Respones Matching", IEEE Trans Automat. Contr., Vol. AC-31, No.1, Jan. 1987, pp.59-61

28. Pal, J., "Improved Pade approximants using stability equation method", Electron. Lett., Vol.19, No.11, May. 1983, pp.426-427

29. Parthasarathy, R. and Jayasimha, K.N., "LINEAR SYSTEM REDUCTION BY CAUER CONTINUED - FRACTION EXPANTION ABOUT s = a AND s = ∞ ALTERNATELY", Electron. Lett., Vol.18, No.9, Apr. 1982, pp.244-246

30. Philip D. Olivier, "On the Relationship Between the Model Order Reduction Problem and the Simultaneous Stabilization Problem", IEEE Trans Automat. Contr., Vol. AC-32, No.1 Jan. 1987, pp. 54-55

31. Rao, A.S., "On Linear System Reduction by Impulse Energy Approximation", IEEE Trans Automat. Contr., Vol. AC-31, No.6, Jun.1986, pp.551-552

32. Robert E. Skelton, "DYNAMIC SYSTEMS CONTROL LINEAR SYSTEMS ANALYSIS AND SYNTHESIS", John Wiley & Sons, SINGAPORE, 1988

33. Sandell, N. R., Jr., Varaiya, P., Athans, M. and Safonov, M. G., "Survey of Decentralized Control Methods for Large Scale Systems", IEEE Trans Automat. Contr., Vol. AC-23, No.2, Apr.1978 pp.108-128

34. Shamash, Y., "Stable Reduced-Order Models Using Pade - Type Approximations", IEEE Trans Automat. Contr., Oct. 1974,

pp.615-616

35. Shamash, Y., "Multivariable system reduction via modal methods and Pade's Approximations", IEEE Trans Automat. Contr., 1975, pp.815-817

36. Wuytack, L., "APPLICATIONS OF PADE APPROXIMATION IN NUMERICAL ANALYSIS", Lecture Notes in Mathematics Approximation Theory, Dold A. and Eckmann B. editer, Springer - Verlag, New York 1976 pp.453-466

ภาคผนวก

ภาคผนวก ก.

โปแกรมที่ใช้ทดสอบระบบในโดเมนของเวลา

```
/*****************************************************************
 *                                                               *
 *        THIS  PROGRAM  IS  ANALYSIS  REDUCED  ORDER            *
 *        MODEL  IN  TIME  DOMAIN  OF  LINEAR  SYSTEM            *
 *             BY  USING  SINGULAR  PERTURBATION                 *
 *                                                               *
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "print.h"
#include "initial.h"
#include "swap.h"
#include "fimatrix.h"

main()
{
 float   A[ROWMAX][COLMAX];
 int     row_A, col_A;
 float   B[ROWMAX][COLMAX];
 int     row_B, col_B;
 float   C[ROWMAX][COLMAX];
 int     row_C, col_C;
 float   W[ROWMAX][COLMAX], D[ROWMAX][COLMAX], P[ROWMAX][COLMAX];
 int     row_W, col_W, row_D, col_D, row_P, col_P;
 float   V[ROWMAX][COLMAX], U[ROWMAX][COLMAX], lamda[MAX];
 int     row_V, col_V;
 float   Diaegn[ROWMAX][COLMAX];
 float   Bm1[ROWMAX][COLMAX], áBm2[ROWMAX][COLMAX], Bm3[ROWMAX][COLMAX];
 int     row_Bm1, col_Bm1, row_Bm2, col_Bm2, árow_Bm3, col_Bm3;
 int     i, j;

/*__________ Variables after partition matrix _____________*/
 float   A11[ROWMAX][COLMAX],   A12[ROWMAX][COLMAX];
 int     row_A11, col_A11,      row_A12, col_A12;
 float   B1[ROWMAX][COLMAX], B2[ROWMAX][COLMAX], B_bar[ROWMAX][COLMAX];
 int     row_B1, col_B1, row_B2, col_B2, row_B_bar, col_B_bar;
 float   C1[ROWMAX][COLMAX], C2[ROWMAX][COLMAX];
 int     row_C1, col_C1,     row_C2, col_C2;
 float   U22[ROWMAX][COLMAX], U21[ROWMAX][COLMAX], IU22[ROWMAX][COLMAX];
 int     row_U22, col_U22, row_U21, col_U21, row_IU22, col_IU22;
 float   Diaegn2[ROWMAX][COLMAX], IDag2[ROWMAX][COLMAX];
 int     row_Dia2, col_Dia2, row_IDag2, col_IDag2;

/*_____________ PARAMETER OF REDUCED ORDER MODEL _____________*/
 float   F[ROWMAX][COLMAX], G[ROWMAX][COLMAX];
```

```
 int     row_F, col_F,       row_G, col_G;
 float   H[ROWMAX][COLMAX], L[ROWMAX][COLMAX];
 int     row_H, col_H,       row_L, col_L;
 FILE    *fp;
 char    name[20];
 int     printer;
 void check(void);     /* stop to inspection data */
 /* print output to printer */
 void PRN_matrix(float matrix[][COLMAX], int row, int col);

/*****************************************************************
 *                                                               *
 *                ONE IS' ON SELECTION OF STATE TO               *
 *                BE RETAINED IN THE REDUCED MODEL               *
 *                                                               *
 *****************************************************************/

 clrscr();
 printf("Do you want output to printer ? (y/n) : ");
 if ( (printer = getchar()) == 'y')
      open_prn();
 (void)getchar();

/*_____________________ INPUT DATA _____________________________*/
 printf("Input external file.\n");
 printf("It save data matrix A, B, C\n");
 printf("Enter file name: ");
 gets(name);
 while ( (fp = fopen(name, "r")) == NULL)
 {
     printf("Can't open the file %s\n", name);
     printf("Try input file name again: ");
     gets(name);
 }

 Fi_Get_matrix(fp, A, &row_A, &col_A);
 Fi_Get_matrix(fp, B, &row_B, &col_B);
 Fi_Get_matrix(fp, C, &row_C, &col_C);
 fclose(fp);
 printf("\n* MATRIX A *\n");
 Print_matrix(A, row_A, col_A);
 if (printer == 'y')
 {
   PRINT(PRN, "\n* MATRIX A *\n\n");
   PRN_matrix(A, row_A, col_A);
   PRINT(PRN, "\n");
 }
 check();
 printf("\n* MATRIX B *\n");
```

```
 Print_matrix(B, row_B, col_B);
 if (printer == 'y')
 {
   PRINT(PRN, "\n* MATRIX B *\n\n");
   PRN_matrix(B, row_B, col_B);
   PRINT(PRN,"\n");
 }
 check();
 printf("\n* MATRIX C *\n");
 Print_matrix(C, row_C, col_C);
 if (printer == 'y')
 {
   PRINT(PRN, "\n* MATRIX C *\n\n");
   PRN_matrix(C, row_C, col_C);
   PRINT(PRN,"\n");
 }
 check();
/*___________ CALCULATION EIGENVALUES OF MATRIX A _____________*/
 printf("* EIGENVALUES OF MATRIX A *\n");
 if ( (Eigen_QR(A, row_A, col_A, lamda)) != 1)
 {
   printf("Eigen value is complex number\n");
   printf("Can't be slove to continue\n");
   printf("\n!!! Return to system now !!!\n");
   exit(1);
 }
 check();
 printf("Eigen values of matrix A before absolute sorting\n");
 printf("\n");
 for (i = 1; i <= row_A; i++)
     printf("lamda[%d]: %f\n", i, lamda[i]);
 printf("\n");
 if (printer == 'y')
 {
    PRINT(PRN, "Eigen values of matrix A before absolute sorting\n");
    for (i = 1; i <= row_A; i++)
       PRINT(PRN, "lamda[%d]: %f\n", i, lamda[i]);
    PRINT(PRN, "\n");
 }

/*_________________ CHECK STABLE MATRIX A ____________________*/
 for (i = 1; i <= row_A; i++)
   if (lamda[i] > 0)
   {
      printf("Eigen values of matrix A is positive number\n");
      printf("lamda[%d] = %f\n", i, lamda[i]);
      printf("Matrix A is unstable matrix\n");
      exit(2);
   }
```

```
/*_____________ CHECK JORDAN CANONICAL FROM _____________*/
 for (i = 1; i <= row_A-1; i++)
 {
     for (j = i+1; j <= row_A; j++)
       if (lamda[i] == lamda[j])
       {
          printf("There is same eigenvalues lamda[%d] = lamda[%d])", i, j);
          printf(" = %f)\n", lamda[j]);
          printf("Tranform matrix A to be Jordan canonical form\n");
          printf("Can't be slove\n");
          exit(3);
       }
 }

/*_____________ SORTING ABSOLUTE EIGENVALUES _____________*/
 for (i = 1; i <= row_A-1; i++)
 {
    for (j = i+1; j <= row_A; j++)
    {
       if ( fabs(lamda[i]) > fabs(lamda[j]) )
       fswap(&lamda[i], &lamda[j]);
    }
 }
 printf("Eigen values of matrix A after absolute values sorting\n");
 Print_colvector(lamda, row_A);
 if (printer == 'y')
 {
   PRINT(PRN,"Eigen values of matrix A after absolute sorting\n");
   for (i = 1; i <= row_A; i++)
        PRINT(PRN, "lamda[%d] : %f\n", i, lamda[i]);
   PRINT(PRN, "\n");
 }

/*_____________ INITIAL DIAGONAL EIGENVALUES MATRIX _____________*/
 for (i = 1; i <= row_A; i++)
 {
    for (j = 1; j <= col_A; j++)
         Diaegn[i][j] = 0.0;
    Diaegn[i][i] = lamda[i];
 }
 printf("\n* DIAGONAL EIGENVALUES OF MATRIX A *\n");
 Print_matrix(Diaegn, row_A, col_A);
 if (printer == 'y')
 {
    PRINT(PRN,"\n* DIAGONAL EIGENVALUES OF MATRIX A *\n");
    PRN_matrix(Diaegn, row_A, col_A);
    PRINT(PRN,"\n");
 }
```

```
 check();
/*__________ CALCULATION EIGENVECTORS OF MATRIX A _____________*/
 printf("eigen values and eigenvector of matrix A\n");
 Jacobi_egn(A, row_A, col_A, lamda, V, &row_V, &col_V);
 check();

/*_____________ FIND RIGHT & LEFT EIGENVECTORS _________________*/
 printf("* RIGHT EIGENVECTORS ( V ) *\n");
 Print_matrix(V, row_V, col_V);
 Inverse_matrix(V, row_V, col_V, U);
 printf("\n* LEFT EIGENVECTOR ( U ) *\n");
 Print_matrix(U, row_A, col_A);
 if (printer == 'y')
 {
   PRINT(PRN, "\n* RIGHT EIGENVECTOR OF MATRIX A ( V ) *\n");
   PRN_matrix(V, row_V, col_V);
   PRINT(PRN, "\n");
   PRINT(PRN, "\n* LEFT EIGENVECTOR OF MATRIX A ( U ) *\n");
   PRN_matrix(U, row_A, col_A);
   PRINT(PRN, "\n");
 }
 check();
/*__ READ W FROM EXTERNAL FILE (W[i,j] = INTEGRAL X[i]*X[j] ___*/
 printf("* STATE ENERGY PARTICIPATION MATRIX ( W ) *\n");
 printf("              ⌠ ");
 printf("\n W[i,j] = |  (x[i]*x[j]) dt\n");
 printf("              ⌡\n");
 printf("W is already calculation and it's in external file.\n");
 printf("Now read matrix W from that file.\n");
 printf("\nINPUT MATRIX W\n");
 printf("enter file name: ");
 gets(name);
 while ( (fp = fopen(name, "r")) == NULL)
 {
     printf("Can't open the file %s\n", name);
     printf("Try input file name again: ");
     gets(name);
 }
 Fi_Get_matrix(fp, W, &row_W, &col_W);
 fclose(fp);
 printf("              ⌠ ");
 printf("\n W[i,j] = |  (x[i]*x[j]) dt\n");
 printf("              ⌡\n");
 printf("\n* MATRIX W *\n");
 Print_matrix(W, row_W, col_W);
 if (printer == 'y')
 {
   PRINT(PRN, "\n* STATE ENERGY PARTICIPATION MATRIX ( W ) *\n");
   PRINT(PRN,"\n");
```

```
   PRINT(PRN,"           ⌠ ");
   PRINT(PRN,"\n W[i,j] = ⌡ (x[i]*x[j]) dt\n");
   PRINT(PRN,"           ⌡\n");
   PRINT(PRN, "\n* MATRIX W *\n");
   PRN_matrix(W, row_W, col_W);
   PRINT(PRN, "\n");
 }
 check();

/*_____________ CALCULATION D = C_transpose * C _____________*/
 printf(" D = C_transpose * C\n\n");
 row_Bm1 = col_C;
 col_Bm1 = row_C;
 for (i = 1; i <= row_Bm1; i++)
 {
    for (j = 1; j <= col_Bm1; j++)
        Bm1[i][j] = C[j][i];   /* Bm1 = transpose of matrix C */
 }
 Mul_matrix(Bm1, row_Bm1, col_Bm1, C, row_C, col_C,
            D, &row_D, &col_D);
 printf("* MATRIX D *\n");
 Print_matrix(D, row_D, col_D);
 if (printer == 'y')
 {
    PRINT(PRN, "\n* MATRIX D *\n");
    PRN_matrix(D, row_D, col_D);
    PRINT(PRN, "\n");
 }
 check();
/*_____________ CALCULATION MATRIX P = D*W _____________*/
 printf("MATRIX P = D*W\n");
 printf("* MATRIX P *\n");
 Mul_matrix(D, row_D, col_D, W, row_W, col_W, P, &row_P, &col_P);
 Print_matrix(P, row_P, col_P);
 printf("\nSelect state to be retained in REDUCED ORDER MODEL\n");
 printf("then partition matrice to calculation parameters of ");
 printf("REDUCED ORDER MODEL\n");
 if (printer == 'y')
 {
    PRINT(PRN, "\n* MATRIX P *\n");
    PRN_matrix(P, row_P, col_P);
    PRINT(PRN, "\n\n");
    PRINT(PRN,"\nSelect state to be retained in REDUCED ORDER MODEL\n");
    PRINT(PRN,"then partition matrice to calculation parameters of ");
    PRINT(PRN,"REDUCED ORDER MODEL\n");
 }
 check();
```

```
/*****************************************************************
 *                                                               *
 *         CALCULATION PARAMETERS OF REDUCED ORDER MODEL         *
 *                                                               *
 *****************************************************************/


/*_________________ READ DATA AFTER PARTITION _________________*/
printf("Partition all matrice and write in external file\n");
printf("enter file name: ");
gets(name);
while ( (fp = fopen(name, "r")) == NULL)
{
    printf("Can't open the file %s\n", name);
    printf("Try input file name again: ");
    gets(name);
}
Fi_Get_matrix(fp, A11, &row_A11, &col_A11);
Fi_Get_matrix(fp, A12, &row_A12, &col_A12);
Fi_Get_matrix(fp, B1, &row_B1, &col_B1);
Fi_Get_matrix(fp, B2, &row_B2, &col_B2);
Fi_Get_matrix(fp, C1, &row_C1, &col_C1);
Fi_Get_matrix(fp, C2, &row_C2, &col_C2);
Fi_Get_matrix(fp, U21, &row_U21, &col_U21);
Fi_Get_matrix(fp, U22, &row_U22, &col_U22);
Fi_Get_matrix(fp, Diaegn2, &row_Dia2, &col_Dia2);
fclose(fp);
printf("\n* MATRIX A11 *\n");
Print_matrix(A11, row_A11, col_A11);
printf("\n* MATRIX A12 *\n");
Print_matrix(A12, row_A12, col_A12);
if (printer == 'y')
{
  PRINT(PRN, "\n* MATRIX A11 *\n");
  PRN_matrix(A11, row_A11, col_A11);
  PRINT(PRN, "\n");
  PRINT(PRN, "\n* MATRIX A12 *\n");
  PRN_matrix(A12, row_A12, col_A12);
  PRINT(PRN, "\n");
}
check();
printf("\n* MATRIX B1 *\n");
Print_matrix(B1, row_B1, col_B1);
printf("\n* MATRIX B2 *\n");
Print_matrix(B2, row_B2, col_B2);
if (printer == 'y')
{
  PRINT(PRN, "\n* MATRIX B1 *\n");
  PRN_matrix(B1, row_B1, col_B1);
```

```
    PRINT(PRN, "\n");
    PRINT(PRN, "\n* MATRIX B2 *\n");
    PRN_matrix(B2, row_B2, col_B2);
    PRINT(PRN, "\n");
  }
  check();
  printf("\n* MATRIX C1 *\n");
  Print_matrix(C1, row_C1, col_C1);
  printf("\n* MATRIX C2 *\n");
  Print_matrix(C2, row_C2, col_C2);
  if (printer == 'y')
  {
    PRINT(PRN, "\n* MATRIX C1 *\n");
    PRN_matrix(C1, row_C1, col_C1);
    PRINT(PRN, "\n");
    PRINT(PRN, "\n* MATRIX C2 *\n");
    PRN_matrix(C2, row_C2, col_C2);
    PRINT(PRN, "\n");
  }
  check();
  printf("\n* LEFT EIGEN VECTOR U21 *\n");
  Print_matrix(U21, row_U21, col_U21);
  printf("\n* LEFT EIGEN VECTOR U22 *\n");
  Print_matrix(U22, row_U22, col_U22);
  if (printer == 'y')
  {
    PRINT(PRN, "\n* LEFT EIGEN VECTOR U21 *\n");
    PRN_matrix(U21, row_U21, col_U21);
    PRINT(PRN, "\n");
    PRINT(PRN, "\n* LEFT EIGENVECTOR U22 *\n");
    PRN_matrix(U22, row_U22, col_U22);
    PRINT(PRN, "\n");
  }
  check();
  printf("\n* DIAGONAL EIGENVALUE OF MATRIX A (no. 2) *\n");
  Print_matrix(Diaegn2, row_Dia2, col_Dia2);
  if (printer == 'y')
  {
     PRINT(PRN, "\n* DIAGONAL EIGENVALUE OF MATRIX A (no. 2) *\n");
     PRN_matrix(Diaegn2, row_Dia2, col_Dia2);
     PRINT(PRN, "\n");
  }
  check();
/*________ CALCULATION PARAMETER OF REDUCED ORDER MODEL ________*/
  printf("\nCALCULATION PARAMETER OF REDUCED ORDER MODEL\n");
  check();
  printf("\n* PARAMETER F *\n");
  Inverse_matrix(U22, row_U22, col_U22, IU22);
  row_IU22 = row_U22;
```

```
col_IU22 = col_U22;
Mul_matrix(A12, row_A12, col_A12, IU22, row_IU22, col_IU22,
           Bm1, &row_Bm1, &col_Bm1);
Mul_matrix(Bm1, row_Bm1, col_Bm1, U21, row_U21, col_U21,
           Bm2, &row_Bm2, &col_Bm2);
Sub_matrix(A11, row_A11, col_A11, Bm2, row_Bm2, col_Bm2,
           F, &row_F, &col_F);
Print_matrix(F, row_F, col_F);
if (printer == 'y')
{
   PRINT(PRN,"\n");
   PRINT(PRN,"PARAMETER OF REDUCED ORDER MODEL\n");
   PRINT(PRN, "\n* PARAMETER F *\n");
   PRN_matrix(F, row_F, col_F);
   PRINT(PRN, "\n");
}
check();
printf("\n* PARAMETER G *\n");
Mul_matrix(U21, row_U21, col_U21, B1, row_B1, col_B1,
           Bm1, &row_Bm1, &col_Bm1);
Mul_matrix(U22, row_U22, col_U22, B2, row_B2, col_B2,
           Bm2, &row_Bm2, &col_Bm2);
Add_matrix(Bm1, row_Bm1, col_Bm1, Bm2, row_Bm2, col_Bm2,
           B_bar, &row_B_bar, &col_B_bar);
Mul_matrix(A12, row_A12, col_A12, IU22, row_IU22, col_IU22,
           Bm1, &row_Bm1, &col_Bm1);
Inverse_matrix(Diaegn2, row_Dia2, col_Dia2, IDag2);
row_IDag2 = row_Dia2;
col_IDag2 = col_Dia2;
Mul_matrix(Bm1, row_Bm1, col_Bm1, IDag2, row_IDag2, col_IDag2,
           Bm2, &row_Bm2, &col_Bm2);
Mul_matrix(Bm2, row_Bm2, col_Bm2, B_bar, row_B_bar, col_B_bar,
           Bm3, &row_Bm3, &col_Bm3);
Sub_matrix(B1, row_B1, col_B1, Bm3, row_Bm3, col_Bm3,
           G, &row_G, &col_G);
Print_matrix(G, row_G, col_G);
if (printer == 'y')
{
   PRINT(PRN, "\n* PARAMETER G *\n");
   PRN_matrix(G, row_G, col_G);
   PRINT(PRN, "\n");
}
check();
printf("\n* PARAMETER H *\n");
Mul_matrix(C2, row_C2, col_C2, IU22, row_IU22, col_IU22,
           Bm1, &row_Bm1, &col_Bm1);
Mul_matrix(Bm1, row_Bm1, col_Bm2, U21, row_U21, col_U21,
           Bm2, &row_Bm2, &col_Bm2);
Sub_matrix(C1, row_C1, col_C1, Bm2, row_Bm2, col_Bm2,
```

```
            H, &row_H, &col_H);
 Print_matrix(H, row_H, col_H);
 if (printer == 'y')
 {
    PRINT(PRN, "\n* PARAMETER H *\n");
    PRN_matrix(H, row_H, col_H);
    PRINT(PRN, "\n");
 }
 check();
 printf("\n* PARAMETER L *\n");
 Mul_matrix(C2, row_C2, col_C2, IU22, row_IU22, col_IU22,
            Bm1, &row_Bm1, &col_Bm1);
 Mul_matrix(Bm1, row_Bm1, col_Bm1, IDag2, row_IDag2, col_IDag2,
            Bm2, &row_Bm2, &col_Bm2);
 Mul_matrix(Bm2, row_Bm2, col_Bm2, B_bar, row_B_bar, col_B_bar,
            L, &row_L, &col_L);
 for (i = 1; i <= row_L; i++)
 {
     for (j = 1; j <= col_L; j++)
         L[i][j] = -L[i][j];
 }
 Print_matrix(L, row_L, col_L);
 if (printer == 'y')
 {
     PRINT(PRN, "\n* PARAMETER L *\n");
     PRN_matrix(L, row_L, col_L);
     PRINT(PRN, "\n");
 }
 printf("\n\nREDUCED ORDER MODEL.....COMPLETE\n");
 if (printer == 'y')
 {
    PRINT(PRN, "\n\n\nREDUCED ORDER MODEL.....COMPLETE\n");
    clos_prn();
 }
 gotoxy(24,24);
 printf("press any key to systems.");
 getch();
 clrscr();
return 0;
}

/*****************************************************************
 *        THIS FUNCTION IS  STOP TO CHECK CALCULATION            *
 *****************************************************************/
void check(void)      /* stop to inspection data */
{
 char *text = "press spacebar to continue: ";
 gotoxy(24,24);
```

```
 printf("%s", text);
 while ( getch() != ' ')
  ;
 clrscr();
}

/*****************************************************************
 *              FUNCTION  INPUT MATRIX FROM FILE                 *
 *****************************************************************/
Fi_Get_matrix(FILE *fp, float Matrix[][COLMAX], int *row, int *col)
{
  int  i, j;
  char buff[20];

      fscanf(fp,"%d %d\n", row, col);
      for(i = 1; i <= *row; i++)
          for( j = 1; j <= *col; j++)
         {
              fgets(buff, sizeof(buff), fp);
              Matrix[i][j] = atof(buff);
         }
}
/*****************************************************************
 *                FUNCTION PRINT MATRIX  TO SCREEN               *
 *****************************************************************/
void Print_matrix(float Matrix[][COLMAX],int row,int colum)
{
 int i, j;

 printf("\n");
 printf("┌");
 for (i = 0; i < colum * 11; i++)
     printf(" ");
 printf("┐\n");
 for (i = 1; i <= row; i++)
 {
     printf("│");
     for (j = 1; j <= colum; j++)
         printf("%10.4f ",Matrix[i][j]);
     printf("│\n");
 }
 printf("└");
 for (i = 0; i < colum * 11; i++)
      printf(" ");
 printf("┘\n");
}

/*****************************************************************
 *                FUNCTION PRINT MATRIX TO PRINTER               *
```

```
 ************************************************************/
void PRN_matrix(float Matrix[][COLMAX],int row,int colum)
{
 int i, j;
 PRINT(PRN, "\n");
 PRINT(PRN, "┌");
 for (i = 0; i < colum * 11; i++)
      PRINT(PRN, " ");
 PRINT(PRN, "┐\n");
 for (i = 1; i <= row; i++)
 {
     PRINT(PRN, "│");
     for (j = 1; j <= colum; j++)
          PRINT(PRN, "%10.4f ",Matrix[i][j]);
     PRINT(PRN, "│\n");
  }
 PRINT(PRN, "└");
 for (i = 0; i < colum * 11; i++)
      PRINT(PRN, " ");
 PRINT(PRN, "┘\n");
}

/************************************************************
 *                    FUNCTION ADD MATRIX                   *
 ************************************************************/
void Add_matrix(float m1[][COLMAX], int row1, int col1,
                float m2[][COLMAX], int row2, int col2,
                float m3[][COLMAX], int *row3, int *col3)
{
  int i, j;

  if (row1 != row2 || col1 != col2)
  {
     printf("ERROR !!! can't sub matrix\n");
     exit(1);
  }
  *row3 = row1;
  *col3 = col1;
  for (i = 1; i <= row1; i++)
  {
     for (j = 1; j <= col1; j++)
        m3[i][j] = m1[i][j] + m2[i][j];
  }
}

/************************************************************
 *                    FUNCTION SUB MATRIX                   *
 ************************************************************/
void Sub_matrix(float m1[][COLMAX], int row1, int col1,
```

```
                float m2[][COLMAX], int row2, int col2,
                float m3[][COLMAX], int *row3, int *col3)
{
  int i, j;

  if (row1 != row2 || col1 != col2)
  {
     printf("ERROR !!! can't sub matrix\n");
     exit(1);
  }
  *row3 = row1;
  *col3 = col1;
  for (i = 1; i <= row1; i++)
  {
     for (j = 1; j <= col1; j++)
       m3[i][j] = m1[i][j] - m2[i][j];
  }
}

/*************************************************************
 *                  FUNCTION MULTIPLICATION MATRIX           *
 *************************************************************/
void Mul_matrix(float m1[][COLMAX], int row1, int col1,
                float m2[][COLMAX], int row2, int col2,
                float m3[][COLMAX], int *row3, int *col3)
{
 int   row, col, dummy;

 if( col1 != row2)
 {
   printf("ERROR !!!! Unable multiple matrix\n");
   exit(1);
 }
   for ( row = 1; row <= row1; row++)
     for( col = 1; col <= col2; col++)
      {
         m3[row][col] = 0.0;
         for( dummy = 1; dummy <= col1; dummy++)
         m3[row][col] += m1[row][dummy]*m2[dummy][col];
      }
   *row3 = row1;
   *col3 = col2;
}

/*************************************************************
 *          DETERMINANT BY GAUSSIAN ELIMINATION METHOD       *
 *************************************************************/
float Gauss_det(float Matrix[][COLMAX], int row, int col)
{
```

```
 float BUF[ROWMAX][COLMAX];
 int    ID, IW;
 float  W, P;
 int    i, j, k;

 if ( row != col )
  {
    printf(" ERROR !!!! Nonsquare matrix\n");
    exit(1);
  }
 for ( i = 1; i <= row; i++)
   for ( j = 1; j <= col; j++)
      BUF[i][j] = Matrix[i][j];
 ID = 1;
 for ( k = 1; k <= row-1; k++)
  {
     W  = fabs(BUF[k][k]);
     IW = k;
     for ( i = k+1; i <= row; i++)
     {
        if ( W < fabs(BUF[i][k]))
       {
           IW = i;
            W = fabs(BUF[i][k]);
       }
     } /** i LOOP **/
    if ( W != 0 )
     {
       if ( IW != k );
       {
          ID *= -1;
          for ( j = k; j <= row; j++)
               fswap (&BUF[k][j], &BUF[IW][j]);
       }
       P = BUF[k][k];
       for ( i = k+1; i <= row; i++)
       {
           W = BUF[i][k]/P;
           for ( j = k+1; j <= row; j++)
                BUF[i][j] -= W*BUF[k][j];
       }
   } /** end k LOOP **/
   W = (float)ID;
   for ( i = 1; i <= row; i++ )
       W *= BUF[i][i];
 } /** end if W != 0 **/
return W;
}
```

```
/*****************************************************************
 *      INVERSE MATRIX BY GAUSS - JORDAN ELIMINATION METHOD      *
 *****************************************************************/
void Inverse_matrix(float Matrix[][COLMAX], int row, int col,
                    float Ivre[][COLMAX])
{
 int ID[ROWMAX];
 float P;
 int M;
 int i, j, k, k1;

 if ( row != col)
  {
    printf("ERROR !!!!  Nonsquare matrix\n");
    exit(1);
  }
 for (i = 1; i <= row; i++)
  {
   ID[i] = i;
   for(j = 1; j <= col; j++)
      Ivre[i][j] = Matrix[i][j];
  }
 for (k = 1; k <= row; k++)
  {
   P = fabs(Ivre[k][k]);
   M = k;
   for(j = k; j <= row; j++)
      if (P < fabs(Ivre[j][k]))
      {
         P = fabs(Ivre[j][k]);
         M = k;
      }
   if (P == 0)
    {
      printf("Can't be solved");
      exit(1);
    }
   if (M != k)
    {
       for (j = 1; j <= row; j++)
           fswap(&Ivre[k][j], &Ivre[M][j]);
       swap(&ID[k], &ID[M]);
    }
   P = Ivre[k][k];
   Ivre[k][k] = 1;
   for (j = 1; j <= row; j++)
       Ivre[k][j] /= P;
   for (i = 1; i <= row; i++)
     if (i != k)
```

```
      {
        for (j = 1; j <= row; j++)
            if (j != k)
               Ivre[i][j] -= Ivre[i][k]*Ivre[k][j];
       Ivre[i][k] /= -P;
      }
  } /* end k loop */
 for (j = 1; j <= row-1; j++)
  {
   if (j != ID[j])
    {
     for (k = j+1; k <= row; k++)
       if (j == ID[k])
          k1 = k;
     for (i = 1; i <= row; i++)
         fswap(&Ivre[i][j], &Ivre[i][k1]);
    }
  }
}

/*****************************************************************
 *      QR METHOD WITH SHIFT REAL AND COMPLEX EIGENVALUES        *
 *****************************************************************/
int Eigen_QR (float Matrix[][COLMAX], int row, int col,
              float lamda[])
{
 float A[ROWMAX][COLMAX], U[ROWMAX], V[ROWMAX], W[ROWMAX];
 float BUF[ROWMAX][COLMAX];
 float C, D, P, Q, R, S, T;
 int   N;
 int   i, j, r;
 int   count = 1;          /*  set count eigen value */
 int   tstcmplx = 1;

 if (row != col)
  {
   printf("ERROR !!!! Can't be solved. It's nonsquare matrix\n");
   exit(1);
  }
 for (i = 1; i <= row; i++)
     for (j = 1; j <= row; j++)
         A[i-1][j-1] = Matrix[i][j];
 N = row-1;

/*_____________________ HOUSEHOLDER REDUCTION ____________________*/
 for (r = 0; r <= N-2; r++)
 {
     W[r] = 0.0; P = 0.0;
     for (i = r+1; i <= N; i++)
```

```
        P += A[i][r]*A[i][r];
    S = sqrt(P);
    if (A[r+1][r] < 0)
        S = -S;
    if (S != 0)
   {
       W[r+1] = sqrt((1.0+A[r+1][r]/S)/2.0);
       for (i = r+2; i <= N; i++)
           W[i] = A[i][r]/2.0/S/W[r+1];
       T = 0.0;
       for (i = 0; i <= N; i++)
      {
           P = 0.0; Q = 0.0;
           for (j = r+1; j <= N; j++)
          {
               P += A[i][j]*W[j];
               Q += A[j][i]*W[j];
          }
        U[i] = P;
        V[i] = Q;
        T += P*W[i];
      }/* end i loop */
  for (i = 0; i <= N; i++)
     for (j = r; j <= N; j++)
         A[i][j] = A[i][j] -
                   2.0*W[i]*V[j]-2.0*W[j]*U[i]+4.0*T*W[i]*W[j];
    }/* end if S != 0 */
  }/* end r loop */
R = 0.0; D = 0.0;

/*_____________________ SHIFT _____________________*/
do
{
 D += A[N][N];
 for (i = 0; i <= N; i++)
   A[i][i] -= A[N][N];

/*_____________________ QR FACTORISATION _____________________*/
 R += 1; Q = 0.0;
 for (i = 0; i<= N-1; i++)
 {
  if (A[i][i] == 0)
     T = 2.0*atan(1);
  else
     T = atan(A[i+1][i]/A[i][i]);
  C = cos(T);  S  = sin(T);
  U[i] = C;  V[i] = S;
  for (j = i; j <= N; j++)
  {
```

```
      P = A[i][j];
      A[i][j] = C*P+S*A[i+1][j];
      A[i+1][j] = -S*P+C*A[i+1][j];
   }
   W[i] = 0.0;
   if (fabs(T) > 0.000001)
      W[i] = 1.0;
   if (i != 0)
      Q += W[i]*W[i-1];
 }/* end i loop */

/*____________________ FIND PRODUCT RQ _____________________________*/
 for (j = 0; j <= N-1; j++)
 {
     C = U[j]; S = V[j];
     for (i = 0; i <= j+1; i++)
     {
         P = A[i][j];
         A[i][j] = C*P+S*A[i][j+1];
         A[i][j+1] = -S*P+C*A[i][j+1];
     }
 }/* end j loop */

/*____________________ PRINT MATRIX _____________________________*/
/* printf("\nITERATION NO. %f\n", R);
 for (i = 0; i <= N; i++)
  {
   for (j = 0; j <= N; j++)
   {
     printf("%10.4f " , A[i][j]);
   }
   printf("\n");
  } */

} while (Q > 0.0);

/*____________________ PRINT RESULTS _____________________________*/
printf("\nEIGENVALUES ARE:\n");
for (i = 0; i <= N; i++)
{
    if (W[i] != 1)
    {
        printf("%f\n", A[i][i]+D);
        lamda[count++] = A[i][i] + D;
    }
    else
    {
        P = (A[i][i]+A[i+1][i+1])/2.0;
        Q = A[i][i]*A[i+1][i+1]-A[i][i+1]*A[i+1][i];
```

```
        R = P*P-Q;    S = sqrt(fabs(R));    P += D;
        if (R < 0)
        {
           printf("%f + j * %f\n", P, S);
           printf("%f - j * %f\n", P, S);
           tstcmplx = -1;
        }
       else
       {
          printf("%f\n", P+S);
          lamda[count++] = P + S;
          printf("%f\n", P-S);
          lamda[count++] = P - S;
       }
    i++;
   }/* end else of if W[i] != 1 */
  }/* end i loop */
return tstcmplx;
}

/*************************************************************
 *         EIGEN VALUE & EIGEN VECTOR BY JACOBI METHOD       *
 *************************************************************/
int Jacobi_egn  (float Matrix[][COLMAX], int row, int col,
       float lamda[], float v[][COLMAX], int *rowEV, int *colEV)
{
 float  ax, a[ROWMAX][COLMAX];
 int    i, i1, j, j1, kt;
 float  al, sa, ss, s1, s2, s7, s8;

 void Jacobi_print_egn(float a[][COLMAX], float v[][COLMAX],
                       float lamda[], int n, int kt);

 if (row != col)
 {
   printf("ERROR !!!! nonsquare matrix\n");
   exit(1);
 }

/*_________________ INITIAL VALUE EIGEN VECTOR _________________*/
 *rowEV = row;
 *colEV = col;
 for (i = 1; i <= row; i++)
 {
    for (j = 1; j <= col; j++)
    {
       a[i][j] = Matrix[i][j];
       v[i][j] = 0.0;
    }
```

```
    v[i][i] = 1.0;
 }

/*_____________________________________________________________*/
 printf("\nadmissible error: %e\n", EA);
 printf("max. iterations: %d\n", KM);
 for (kt = 1; kt < KM; kt++)
 {
    ax = 0.0;
    for (i = 1; i <= row-1; i++)
    {
        for (j = i+1; j <= row; j++)
            if (fabs(a[i][j]) > ax)
            {
                i1 = i;
                j1 = j;
                ax = a[i][j];
            }
    }/* end for i loop */
   if (ax == 0)
   {
      Jacobi_print_egn(a, v, lamda, row, kt);
      return 1;
   }
   ss = a[i1][i1] - a[j1][j1];
   al = fabs(ss)/2.0;
   sa = sqrt(al*al + a[i1][j1]*a[i1][j1]);
   s8 = 1.0/sqrt(2)*sqrt(1+al/sa);
   s7 = a[i1][j1]/(2*sa*s8);
   if (ss < 0)
      s7 = -s7;
   for (i = 1; i <= row; i++)
   {
       if ((i != i1) && (i != j1))
       {
          s1 = a[i1][i]*s8 + a[j1][i]*s7;
          s2 = a[j1][i]*s8 - a[i1][i]*s7;
          a[i1][i] = s1;
          a[j1][i] = s2;
       }
   }
   s1 = a[i1][i1]*s8*s8 + 2.0*a[i1][j1]*s8*s7 + a[j1][j1]*s7*s7;
   s2 = a[i1][i1]*s7*s7 - 2.0*a[i1][j1]*s8*s7 + a[j1][j1]*s8*s8;
   a[i1][i1] = s1;
   a[j1][j1] = s2;
   a[i1][j1] = 0.0;
   a[j1][i1] = 0.0;
   for (i = 1; i <= row; i++)
   {
```

```
        a[i][i1] = a[i1][i];
        a[i][j1] = a[j1][i];
        s1 = v[i][i1]*s8 + v[i][j1]*s7;
        s2 = v[i][j1]*s8 - v[i][i1]*s7;
        v[i][i1] = s1;
        v[i][j1] = s2;
    }
    if (fabs(ax) <= EA)
    {
       Jacobi_print_egn(a, v, lamda, row, kt);
       return 1;
    }
    if (kt >= KM)
    {
        printf("\nOVER MAXIMUM ITERATION\n");
        exit(2);
    }
 }/* end for kt loop */
 Jacobi_print_egn(a, v, lamda, row, kt);
return 1;
}
/*_____________________ END Jacobi_egn _____________________*/

/*-------- print output of Jacobi_egn program  ----------------*/
void Jacobi_print_egn (float a[][COLMAX], float v[][COLMAX],
                       float lamda[], int n, int kt)
{
 int i,j;

 printf("\nSOLUTION\n");
 printf("eigenvalue\n");
 for (i = 1; i <= n; i++)
 {
     lamda[i] = a[i][i];
     printf("%10.4f ", lamda[i]);
 }
 printf("\n\neigen vector\n");
 Print_matrix(v, n, n);
 printf("\nno. of iterations: %d\n", kt);
}
```

STATEGY.C Monday, June 25, 1990 1:39 pm Page 1

```
/*************************************************
 *    THIS PROGRAM IS CALCULATION STATE ENERGY    *
 *      BY USING ROMBERG INTEGRATION  METHOD      *
 *************************************************/

#include <stdio.h>
#include <math.h>

#define FNF(x)

main()
{
 double EA = 0.00000001;
 double er;
 float a, b, h;
 float S[150], s, s1, s8, s9, sk, sw;
 float rk;
 int   N, i, j, r;

 clrscr();
 printf("* Romberg integration method *\n");
 printf("integral range\n");
 printf("input lower bound: ");
 scanf("%f", &a);
 printf("input upper bound: ");
 scanf("%f", &b);
 printf("input number of partitions (not greather than 130): ");
 scanf("%d", &N);
 printf("no. of partitions: %d\n", N);
 printf("admissible error: %e\n", EA);
 h = b-a; rk = 1; sk = 1;
 s1 = (FNF(a)+FNF(b))/2.0;
 S[0] = h*s1; sw = S[0];
 for (i = 1; i <= N; i++)
 {
    h = h/2.0;
    rk = rk*2.0;
    s = 0.0;
    for (r = 1; r <= rk-1; r++)
    {
       s = s+FNF(a+r*h);
    }
    S[i] = h*(s+s1); er = fabs(1.0 - sw/S[i]); sw = S[i];
    if (er <= EA)
    {
      printf("SOLUTION OF INTEGRAL: %f\n", sw);
      return 1;
    }
 }
```

```
 for (i = 1; i <= N; i++)
 {
    sk = sk*4.0;
    s9 = sk - 1;
    for (j = 0; j <= N - i; j++)
    {
       s8 = S[j+1] - S[j];
       S[j] = S[j+1] + s8/s9;
       er = fabs(1 - sw/S[j]);
       sw = S[j];
       if (er <= EA)
       {
   printf("SOLUTION OF INTEGRAL: %f\n", sw);
   return 1;
       }
    }
 }
 printf("\n* NO CONVERGENCE *\n");

getch();
return -1;
}
```

ภาคผนวก ข.

ตัวอย่างผลลัพธ์ที่ได้จากโปรแกรมในโดเมนของเวลา

```
Do you want output to printer ? (y/n) : y
Input external file.
It saved data of matrix A, B and C.
Enter file name: dat1.dat

* MATRIX A *

[   0.0000    2.0000]
[  -1.0000   -3.0000]

* MATRIX B *

[   2.0000 ]
[  -3.0000 ]

* MATRIX C *

[   3.0000    1.0000]

Eigenvalues of matrix A before absolute sorting
lamda[1]: -1.0000
lamda[2]: -2.0000

Eigenvalues of matrix A after absolute sorting
lamda[1]: -1.0000
lamda[2]: -2.0000

* DIAGONAL EIGENVALUES OF MATRIX A *

[  -1.0000    0.0000]
[   0.0000   -2.0000]

* RIGHT EIGENVECTOR OF MATRIX A ( V ) *

[  -2.0000   -1.0000]
[   1.0000    1.0000]

* LEFT EIGENVECTOR OF MATRIX A ( V ) *

[  -1.0000   -1.0000]
[   1.0000    2.0000]

* STATE ENERGY PARTICIPATION MATRIX ( W ) *
```

$$W[i,j] = \int (x[i]*x[j])\, dt$$

* MATRIX W *

$$\begin{bmatrix} 12.6670 & -1.0000 \\ -1.0000 & 1.8330 \end{bmatrix}$$

* MATRIX D *

$$\begin{bmatrix} 9.0000 & 3.0000 \\ 3.0000 & 1.0000 \end{bmatrix}$$

* MATRIX P *

$$\begin{bmatrix} 111.0030 & -3.5010 \\ 37.0010 & -1.1670 \end{bmatrix}$$

Select state to be retain in REDUCED ORDER MODEL
then partition matrice to calculation parameters
of REDUCED ORDER MODEL

Partition all matrice and write in external file
Enter file name: pdat1.dat

* MATRIX A11 *

$$\begin{bmatrix} 0.0000 \end{bmatrix}$$

* MATRIX A12 *

$$\begin{bmatrix} 2.0000 \end{bmatrix}$$

* MATRIX B1 *

$$\begin{bmatrix} 2.0000 \end{bmatrix}$$

* MATRIX B2 *

[3.0000]

* MATRIX C1 *

[3.0000]

* MATRIX C2 *

[1.0000]

* LEFT EIGENVECTOR U21 *

[1.0000]

* LEFT EIGENVECTOR U22 *

[2.0000]

* DIAGONAL EIGENVALUE OF MATRIX A (No.2) *

[-2.0000]

PARAMETER OF REDUCED MODEL

* PARAMETER F *

[-1.0000]

* PARAMETER G *

[6.0000]

* PARAMETER H *

[2.5000]

```
* PARAMETER L *
[   2.0000]

REDUCED ORDER MODEL ....COMPLETE
```

ภาคผนวก ค.

โปรแกรมที่ใช้ทดสอบระบบในโดเมนของความถี่

```
/*****************************************************
 *   THIS PROGRAM IS COMPUTATION MODEL REDUCTION     *
 *  IN FREQUENCY DOMAIN BY USING PADE APPROXIMATION  *
 *****************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <alloc.h>

#include "swap.h"
#include "d:\sanya\thesis\swap.c"

#define  MAX  20

main()
{
float   a[MAX], b[MAX];
float   c[MAX];
float   d[MAX], e[MAX];

float   C[MAX][MAX], plpole[MAX];

float   pr, pi, mag_pole;

int     i, j, k, m, n;

int     stable;

void Ncoef(float a[], float b[], float c[], int n);
void  Get_vector(float vec[], int num_vec);
void  Gauss(float Matrix[][MAX], int row, int col, float Vector[], int rowV
);
float magnitude_pole(float re, float im);
void  moments(float d[], float e[], int k, float c[]);
void  numer(float a[], float b[], float c[], int n);
int   pole(float coff[], int n);
void  Print_vec(float vec[], int n);


 clrscr();
 printf("Input order of system : ");
 scanf("%d" , &n);
 printf("Input Numerator Coefficients in Ascending Order, 0, 1, 2, ...\n");
 Get_vector(d, n);
 printf("NUMERATOR COEFFICIENTS OF TRANSFER FUNCTION\n");
 for (i = 0; i < n; i++)
     printf("d[%d] = %10.4f\n", i, d[i]);
 printf("Input Denominortor Coefficients in Ascending Order, 0, 1, 2, ...\n
```

```
");
 Get_vector(e, n);
 printf("COEFFICIENT OF DENOMINORTOR OF TRANSFER FUNCTION\n");
 for (i = 0; i < n; i++)
     printf("e[%d] = %10.4f\n", i, e[i]);
 moments(d, e, n, c);
 for (i = 0; i < n; i++)
     printf("time - moment %dth = %10.4f\n", i, c[i]);

 printf("Modification moments value\n");
 for (i = 0; i < n; i++)
 {
     c[i] = pow(-1, i)*fabs(c[i]);
     printf("fitting time - moment %dth = %10.4f\n", i, c[i]);
 }

pole(e, n);


do
{
 printf("Select pole to be retain in model reduction\n");
 printf("REAL PART: ");
 scanf("%f", &pr);
 printf("IMAGINARY PART: ");
 scanf("%f", &pi);
 mag_pole = magnitude_pole(pr, pi);
 printf("MAGNITUDE POLE TO YOUR SELECT IS %f\n", mag_pole);
 printf("Select order to be retain in model reduction : ");
 scanf("%d", &k);
 while ((2*k-1) > n)
 {
    printf("Out off time - moments\n");
    printf("Try again\n");
    printf("Select order to be retain in model reduction : ");
    scanf("%d", &k);
 }

 for (i = 0; i < k  ; i++)
 {
     b[i] = - c[i+1];
     for(j = 0; j < k; j++)
        C[i][j] = c[k+i-j];
 }

 for (i = 0; i <= k; i++)
 {
     plpole[i] = pow(-1,i)*pow(mag_pole,i);
     C[k-1][i] = plpole[i];
```

```
    if (i == k)
       b[k-1] = - plpole[i];
 }

 Gauss(C, k, k, b, k);
 Ncoef(a, b, c, k);

 printf("\tSOLUTION\n\n");
 printf("Order of Model Reduction = %d\n\n", k);
 printf("Numerator Coefficients of Reduced Model\n");
 for (i = 0; i < k; i++)
      printf("Ncoef[%d]: %10.4f\n", i, a[i]);

 printf("\n\n");
 printf("Denominator Coefficients of Reduced Model\n");
 for (i = 0; i < k; i++)
      printf("Dcoef[%d]: %10.4f\n", i, b[i]);

 stable = 1;
 for (i = 0; i < k; i++)
 {
     if (b[i] < 0.0)
     {
         printf("Model Reduction is unstable system !\n");
         printf("Try again\n");
     }
 }
} while(!stable);
getch();
return 0;
}



/*************************************************************
 *                                                           *
 *     THIS IS FUNCTION INPUT VECTORS FROM KEYBOARD          *
 *                                                           *
 *************************************************************/

void Get_vector(float vec[], int num_vec)
{
  int   i;

  for(i = 0; i < num_vec; i++)
  {
      printf("DATA[%d]: ", i);
      scanf("%f", &vec[i]);
  }
```

```
}


/*****************************************************************
 *                                                               *
 *    SLOVE LINEAR EQUATION GAUSSIAN ELIMINATION METHOD          *
 *                                                               *
 *****************************************************************/

void Gauss(float Matrix[][MAX], int row, int col,
           float Vector[], int rowV)
{
 float  A[MAX][MAX], B[MAX];
 int    N;
 float  P, P1, AS;
 int    M;
 int    i, j, k;

 if ((row != col) && (row != rowV))
  {
   printf("Can't be solved. It's nonsquare matrix.\n");
   exit(1);
  }

 N = row;
 for (i = 1; i <= N; i++)
  {
    B[i] = Vector[i-1];
    for ( j = 1; j <= N; j++)
     A[i][j] = Matrix[i-1][j-1];
  }

 for (k = 1; k <= N-1; k++)
  {
    P = fabs(A[k][k]);
    M = k;

    for (i = k; i <= N; i++)
       if (P < fabs(A[i][k]))
        {
          P = fabs(A[i][k]);
          M = i;
        }

    if (P == 0)
     {
      printf("Can't be solved\n");
      exit(1);
     }
```

```
    if (M != k)
     {
       for (j = k; j <= N; j++)
         fswap(&A[k][j], &A[M][j]);
       fswap(&B[k], &B[M]);
     }

    P = A[k][k];
    for (i = k+1; i <= N; i++)
     {
       P1 = A[i][k]/P;
       B[i] -= P1*B[k];
       for (j = k; j <= N; j++)
          A[i][j] -= P1*A[k][j];
     }
  } /* end k loop */

 B[N] /= A[N][N];
 for (i = N-1; i >= 1; i--)
  {
    AS = 0;
    for (j = i+1; j <= N; j++)
       AS += A[i][j]*B[j];
    B[i] = (B[i]-AS)/A[i][i];
  }
 for (i = 1; i <= N; i++)
      Vector[i-1] = B[i];

/*
 printf("\n\nSOLUTION\n");
 for (i = 1; i <= N; i++)
   printf(" X%2d = %g\n", i, B[i]);
*/

}


/*************************** END GAUSS ******************************/


/*****************************************************************
 *                                                               *
 *                    FIND MAGNITUDE OF POLE                     *
 *                                                               *
 *****************************************************************/

float magnitude_pole(float re, float im)
{
```

```
 float x, y, ans, temp;

 x = fabs(re);
 y = fabs(im);

 if (x == 0.0)
    ans = y;
 else if (y == 0.0)
    ans = x;
 else if (x > y)
      {
        temp = y/x;
        ans = x*sqrt(1.0+temp*temp);
      }
 else
 {
    temp = x/y;
    ans = y*sqrt(1.0+temp*temp);
 }

return ans;
}

/********************* END mangitude_pole ***************************/


/*************************************************************************
 *                                                                       *
 *                      FIND TIME - MOMENTS                              *
 *                                                                       *
 *************************************************************************/


void moments(float d[], float e[], int n, float c[])
{
int   i, j;
float temp;

 c[0] = d[0]/e[0];
 for (i = 1; i < n; i++)
 {
     temp = d[i];
     for (j = 0; j < i; j++)
     {
        temp = temp - e[i-j]*c[j];
     }
     c[i] = temp/e[0];
 }
```

```
}

/*********************************************************************
 *                                                                   *
 *                  FIND NUMERATOR OF MODEL REDUCTION                *
 *                                                                   *
 *********************************************************************/

void Ncoef(float a[], float b[], float c[], int n)
{
 int i, j;

 a[0] = b[0]*c[0];
 for (i = 1; i < n; i++)
 {
   a[i] = 0.0;
   for (j = 0; j <= i; j++)
      a[i] = a[i] + b[j]*c[i-j];
 }
}


/*********************************************************************
 *                                                                   *
 *          FIND POLE OF SYSTEMS BY USING BAIRSTOW METHOD            *
 *                                                                   *
 *********************************************************************/

int il, is, mc;
float er, dl, de, dl;
pole(float A[], int N)
{
 float b[MAX], c[MAX], xi[MAX], xr[MAX];
 float ea;
 int km;
 float p, q; /* dl */

 int   i, j, k; /* il */
 int   test;

 int newton(int *k, float A[], float *p,
        float *q, float *ea, int *km, float b[]);
 int quad(float *p, float *q, float xr[], float xi[]);

 ea = 0.0000001;
 km = 200;
 p = 1.0;
 q = -1.0;
 dl = 0.1;
```

```
 A[N] = 1.0;
 printf("* Bairstow method *\n");
 printf("* coefficients *\n");
 for (i = 0; i <= N; i++)
     printf("X[%d] = %f\n", i, A[i]);
 printf("admissible error: %g\n", ea);
 printf("max. iterations: %d\n", km);
 printf("\n\n* solution *\n");
 i1 = 1;
 if (N > 2)
 {
    for (i = N; i >= 3; i -= 2)
    {
         test = newton (&i, A, &p, &q, &ea, &km, b);
         if (test == -1)
             return 1;
         if (km <= 0)
            return 1;
         for (j = 0; j <= (N-2); j++)
             A[j] = b[j];
         test = quad(&p, &q, xr, xi);
         if (test == -1)
            return 1;
    }
 }
 else
    if ((i % 2) == 0)
    {
      p = A[1]/A[0];
      q = A[2]/A[0];
      quad(&p, &q, xr, xi);
      return 1;
    }
 xr[i1] = -A[1]/A[0];
 printf("X %d = %10.3f", i1, xr[i1]);

return 1;
}


/*--------------------  NEWTON  -------------------------*/

int newton(int *k, float A[], float *p,
           float *q, float *ea, int *km, float b[])
{
 int   quit1, quit2;
 int   i;
/* int   is, mc;*/
```

```
 float r, d; /* de, dl;*/
 float c[MAX];

 is = 0;
 mc = 1;
 b[0] = A[0];
 c[0] = A[0];

/* 520 */
 do{
    do{
        b[1] = A[1] - *p*b[0];
        c[1] = b[1] - *p*c[0];
        for (i = 2; i <= *k; i++)
        {
           b[i] = A[i] - *p*b[i-1] - *q*b[i-2];
           c[i] = b[i] - *p*c[i-1] - *q*c[i-2];
        }
        r = c[*k-1] - b[*k-1];
        d = c[*k-2]*c[*k-2] - r*c[*k-3];
        quit1 = 1;
        if (d == 0)
        {
           *p = *p + dl;
           *q = *q + dl;
           quit1 = 0;
        }
    } while (quit1 == 0);
    de = (b[*k-1]*c[*k-2]-b[*k]*c[*k-3])/d;
    dl = (b[*k]*c[*k-2] - b[*k-1]*r)/d;
    *p = *p + de;
    *q = *q + dl;
    if ( (fabs(de) <= *ea) || (fabs(dl) <= *ea) )
    {
       *km = mc;
       return 1;
    }
    quit2 = 1;
    if (mc >= *km)
       is = 1;
    else
    {
        mc = mc+1;
        quit2 = 0;
    }
 }while (quit2 == 0);

return 1;
}
```

```
/*----------------- QUARTRATIC POLYNOMIAL --------------------*/
int quad(float *p, float *q, float xr[], float xi[])
{
  float d2, r1, r2, x1, x2;
  char  *ISSTR, *str;

  *p = *p/2;
  d2 = ((*p) * (*p)) - *q;
  if (d2 >= 0)
  {
     if ( *p <= 0)
        r1 = -(*p) - sqrt(d2);
     else
        r1 = -(*p) + sqrt(d2);
     r2 = *q/r1;
     x1 = 0;
     x2 = 0;
  }
  else
  {
      r1 = -(*p);
      r2 = -(*p);
      x1 = sqrt(-d2);
      x2 = -x1;
  }
  xr[i1] = r1;
  xi[i1] = x1;
  xr[i1+1] = r2;
  xi[i1+1] = x2;

/*--------- PRINT OUTPUT ------------*/

/*-------

 if (is == 1)
    ISSTR = "no convergence";
 else
    ISSTR = itoa(mc, str, 10);
 printf("X %d = %f", i1, xr[i1]);
 if (xi[i1] != 0)
 {
    if (xi[i1] > 0)
        printf(" + ");
    else
        printf(" - ");
 }
 else
```

```
    printf("( %s )\n", ISSTR);
 printf("%10.3f * i ( %s )\n", fabs(xi[i1]), ISSTR); */

printf("%10.4f    %10.4f\n", xr[i1], xi[i1]);

 /*--- 820 ----*/
/* printf("X %d = %10.3f", i1+1, xr[i1+1]);
 if (xi[i1+1] != 0)
 {
    if (xi[i1+1] > 0)
       printf(" + ");
    else
       printf(" - ");

 }
 else
     printf("( %s )\n", ISSTR);

 printf("%10.3f * i ( %s )\n", fabs(xi[i1+1]), ISSTR);

*/

printf("%10.4f    %10.4f\n", xr[i1], xi[i1]);
/*-------- 860 ----------*/

 if (is == 0)
 {
    i1 = i1+1;
    return 1;
 }
 if ( fabs(dl) < fabs(de) )
     er = fabs(dl);
 else
     er = fabs(de);
 printf("\nresidual : %g\n", er);

return -1;
}
```

ภาคผนวก ง.

ตัวอย่างผลลัพธ์ที่ได้จากโปรแกรมในโดเมนของความถี่

```
Input order of system : 6
Input Numerator Coefficients in Ascending Order, 0, 1, 2, ...
DATA[0] = 100000
DATA[1] = 140100
DATA[2] = 6914
DATA[3] = 1469
DATA[4] = 1014
DATA[5] = 1

NUMERATOR COEFFICIENTS OF TRANSFER FUNCTION
d[0] = 100000.0000
d[1] = 140100.0000
d[2] =  6914.0000
d[3] =  1469.0000
d[4] =  1014.0000
d[5] =     1.0000

Input Denominortor Coefficients in Ascending Order, 0, 1, 2, ...
DATA[0] = 1000000
DATA[1] = 2220000
DATA[2] = 1454100
DATA[3] = 248420
DATA[4] = 14541
DATA[5] = 2225
DATA[6] = 1

NUMERATOR COEFFICIENTS OF TRANSFER FUNCTION
e[0] = 1000000.0000
e[1] = 2220000.0000
e[2] = 1454100.0000
e[3] =  248420.0000
e[4] =   14541.0000
e[5] =    2225.0000
e[6] =       1.0000

Fitting Time - Moment  c[0] =   0.1000
Fitting Time - Moment  c[1] =  -0.0819
Fitting Time - Moment  c[2] =   0.1055
Fitting Time - Moment  c[2] =  -0.1260
Fitting Time - Moment  c[4] =   0.1461


Select pole to be retain in model reduction
REAL PART : -100
IMAGINARY PART : 0  .

MAGNITUDE POLE TO YOUR SELECT IS 100.0000

Select order to be retain in model reduction : 2
```

SOLUTION

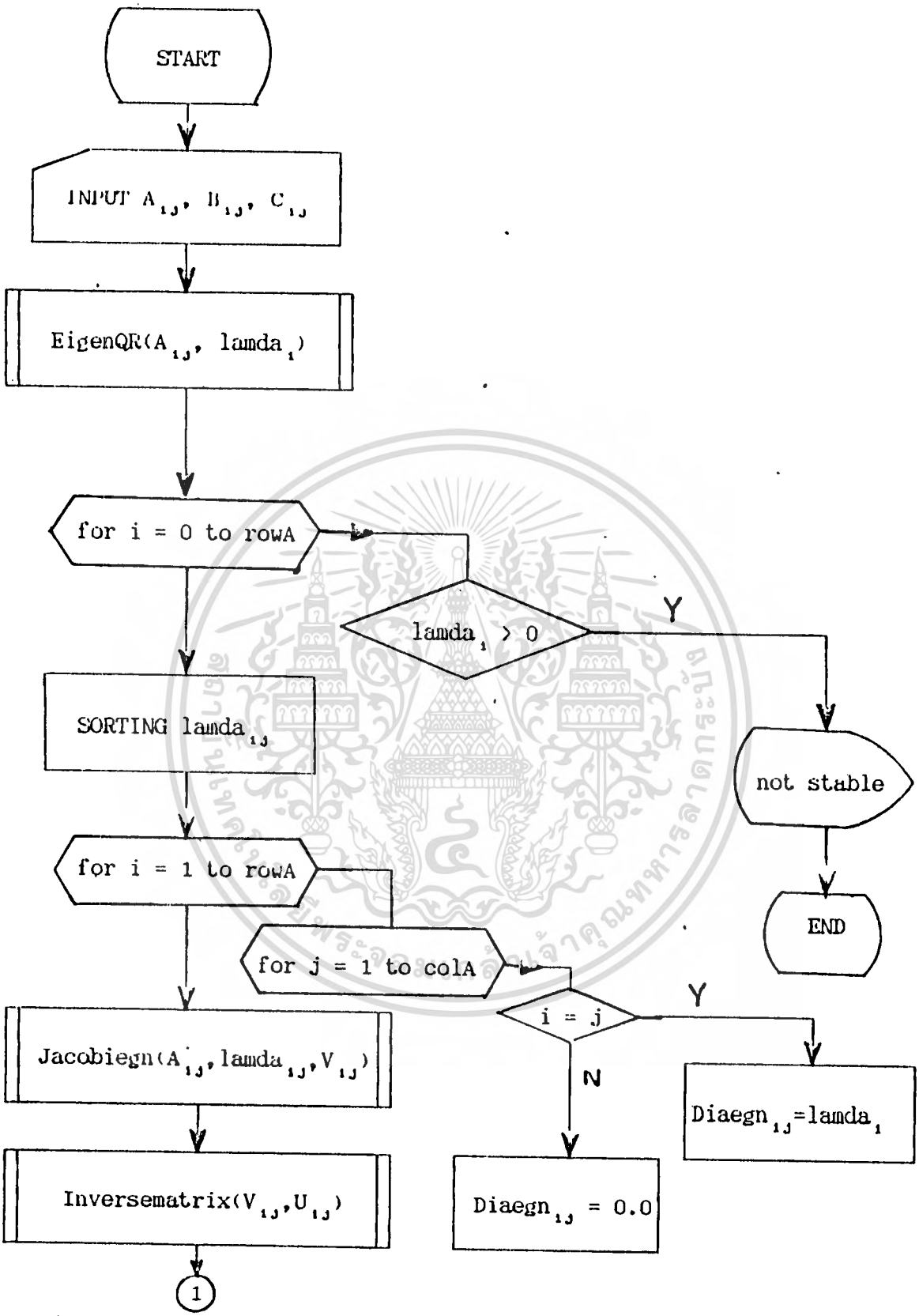
Order of Model Reduction = 2

Numerator Coefficents of Reduced Model
Ncoef[0]: 7.898339
Ncoef[1]: 3.6093

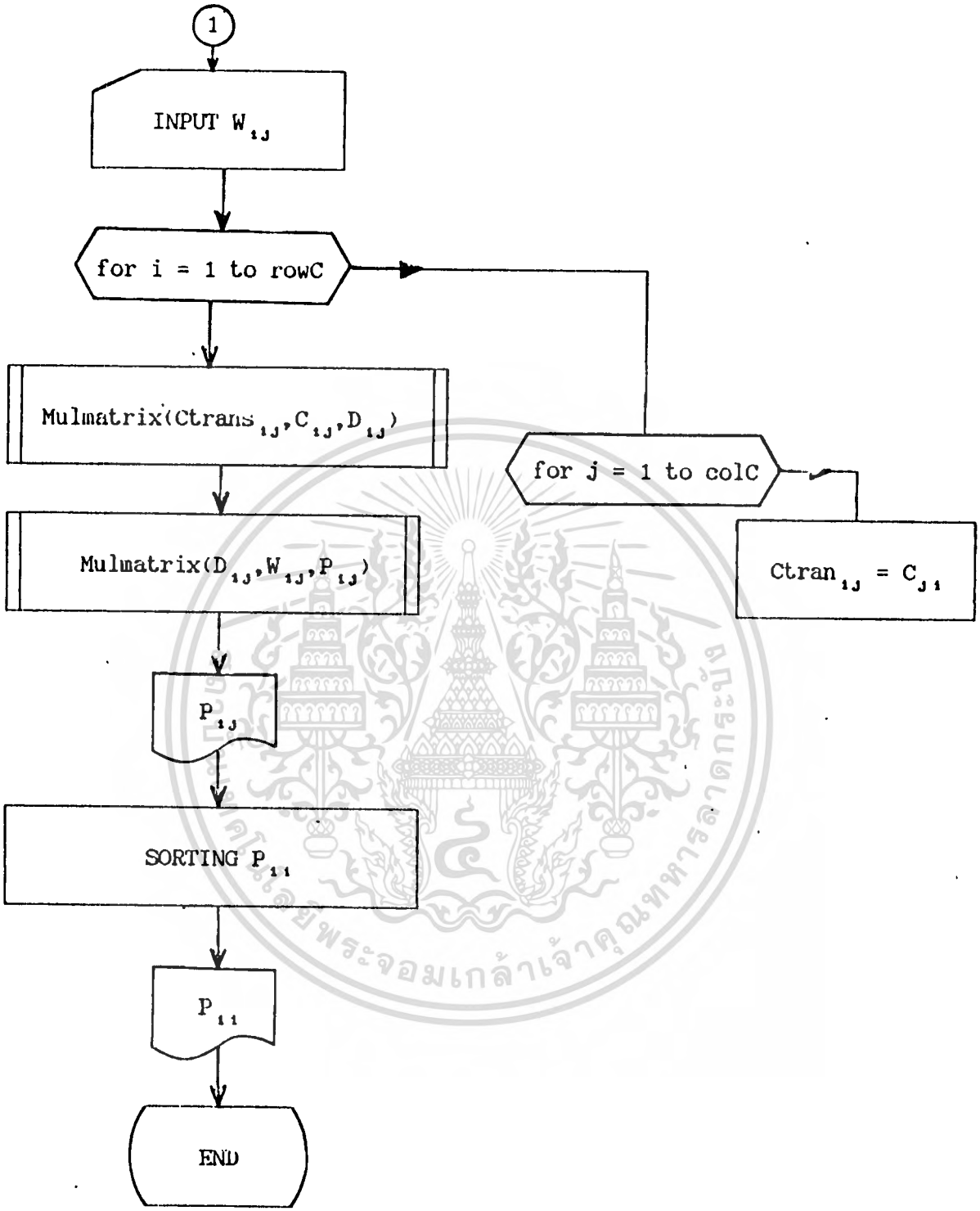
Denominator Coefficents of Reduced Model
Dcoef[0]: 78.983398
Dcoef[1]: 100.789833

ภาคผนวก จ.

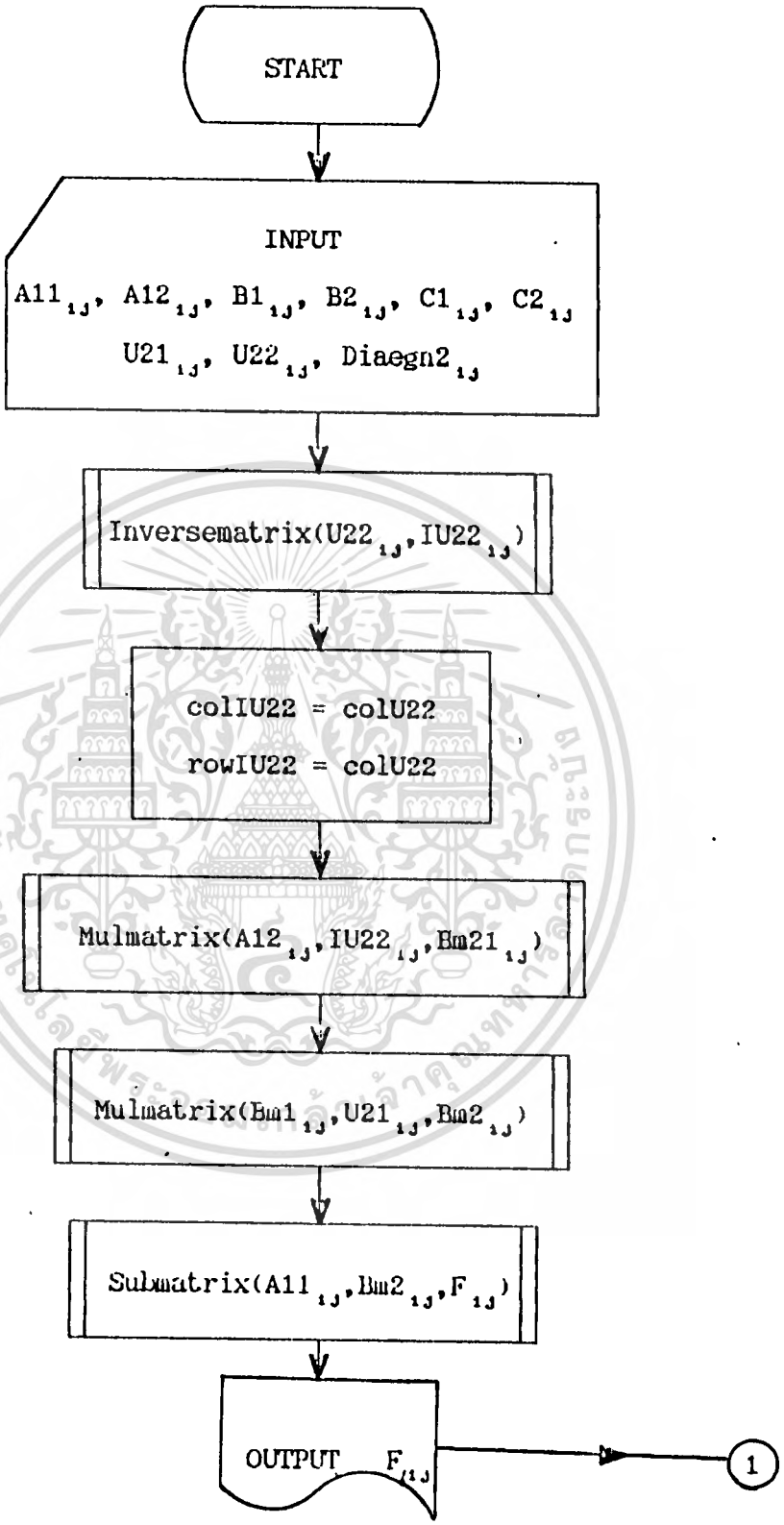
FLOW CHART ที่ใช้ในการออกแบบโปรแกรม



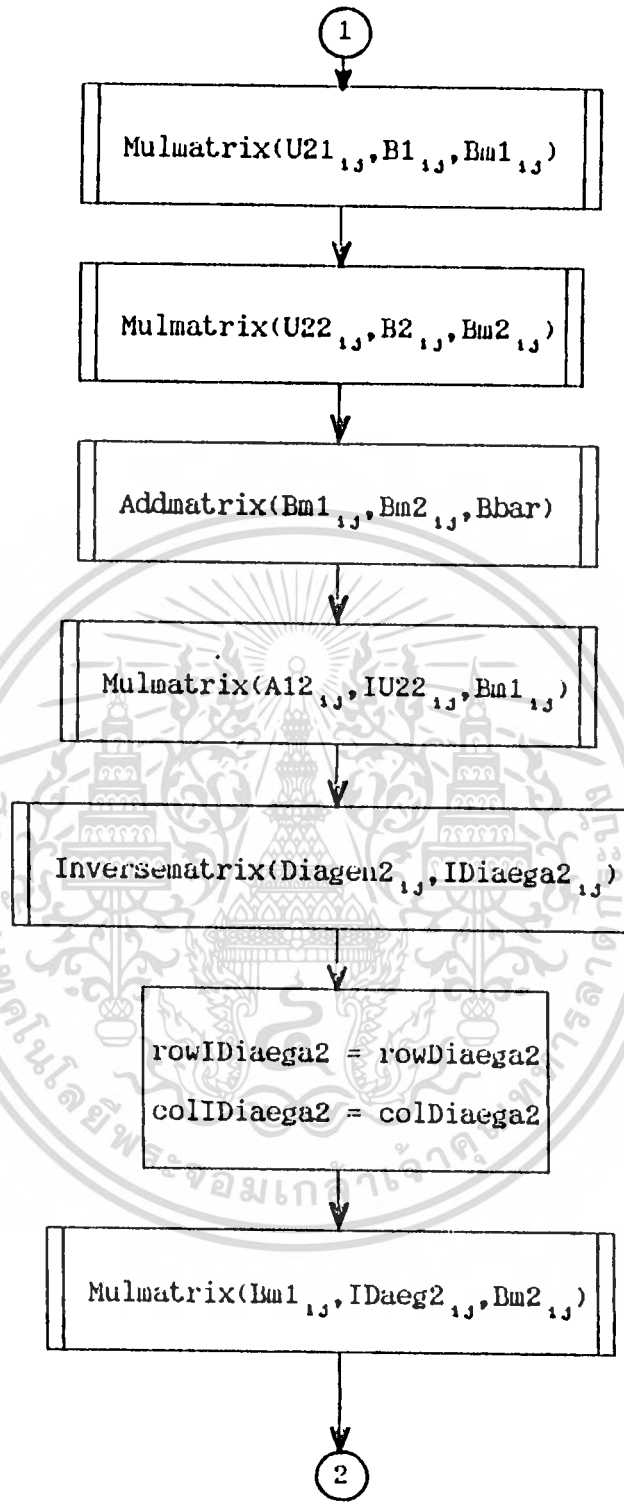
รูปที่ จ.1 FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมของการเลือกสถานะใน DOMAIN ของเวลา



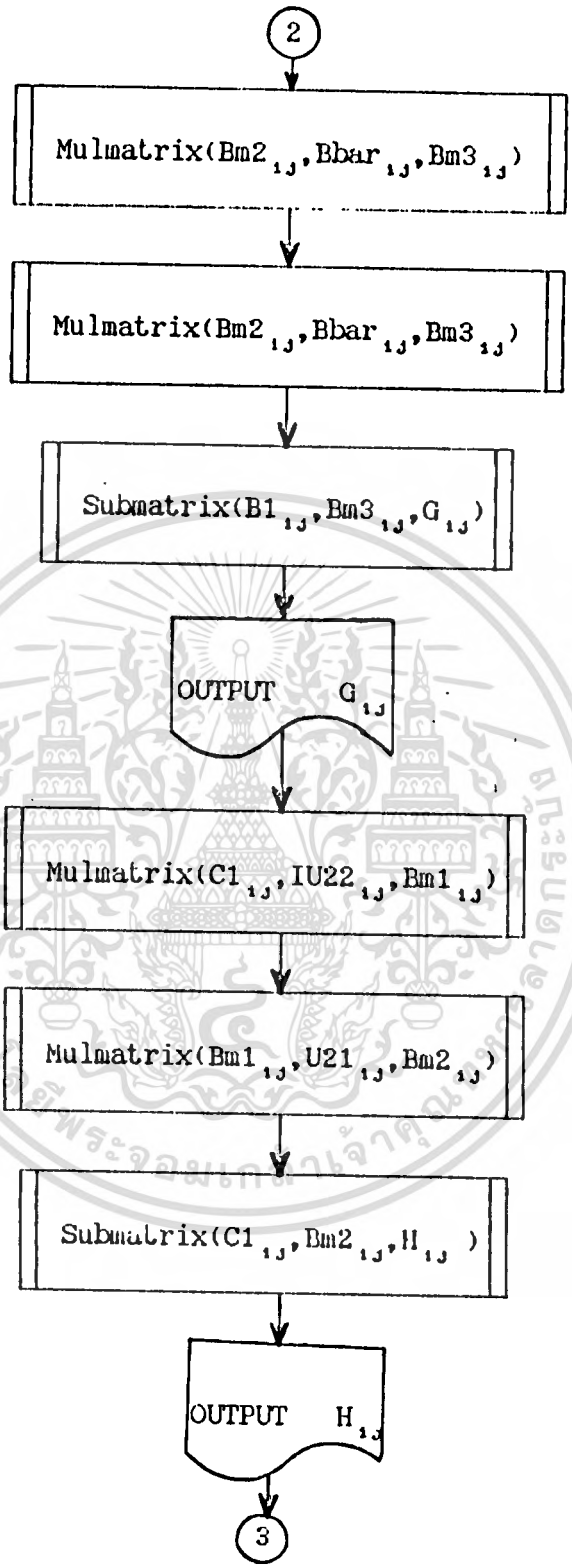
รูปที่ จ.1 (ต่อ) FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมของการเลือกสถานะใน DOMAIN ของเวลา



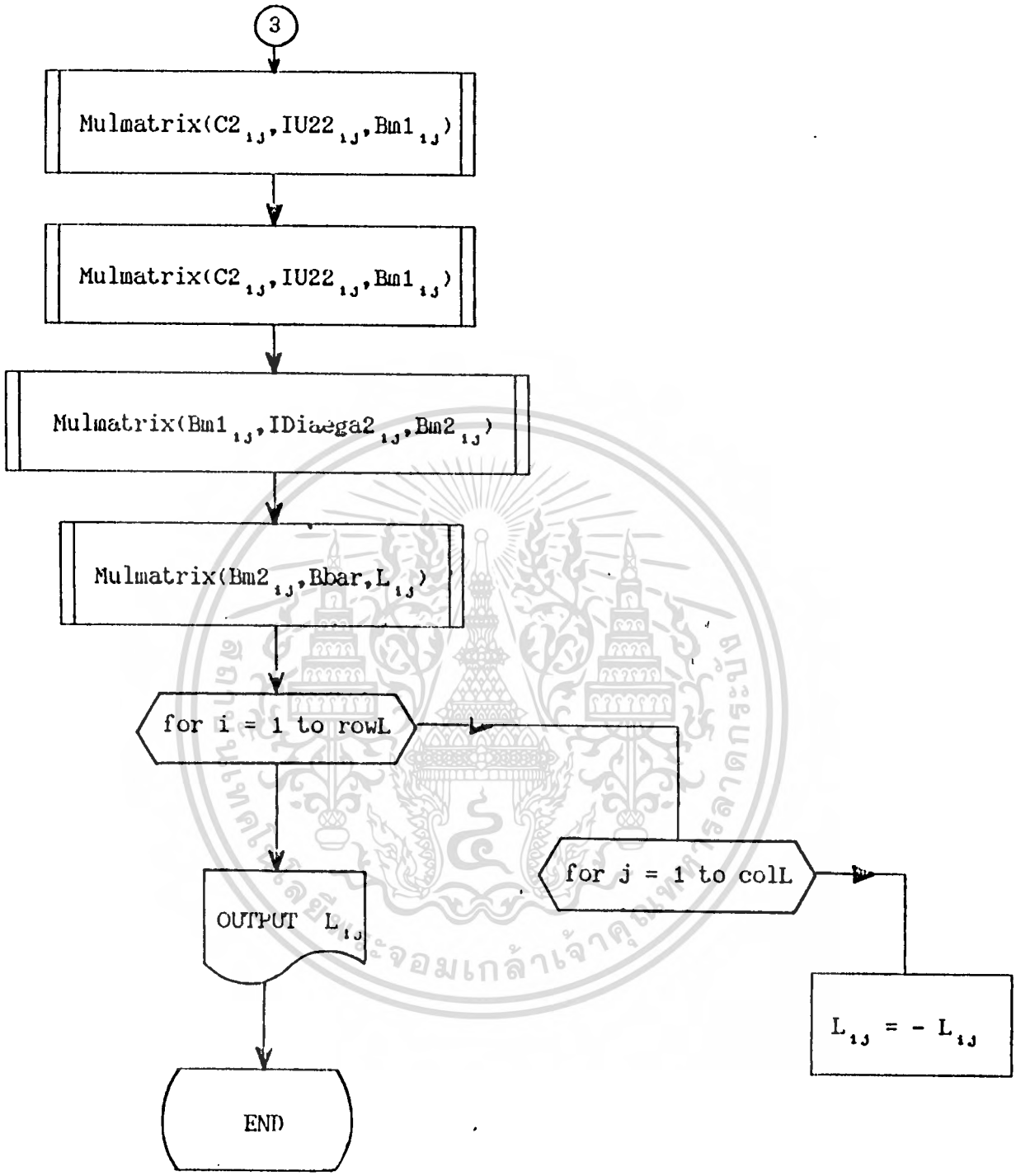
รูปที่ จ.2 FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมการหาพารามิเตอร์ของ MODEL REDUCTION ใน DOMAIN ของเวลา



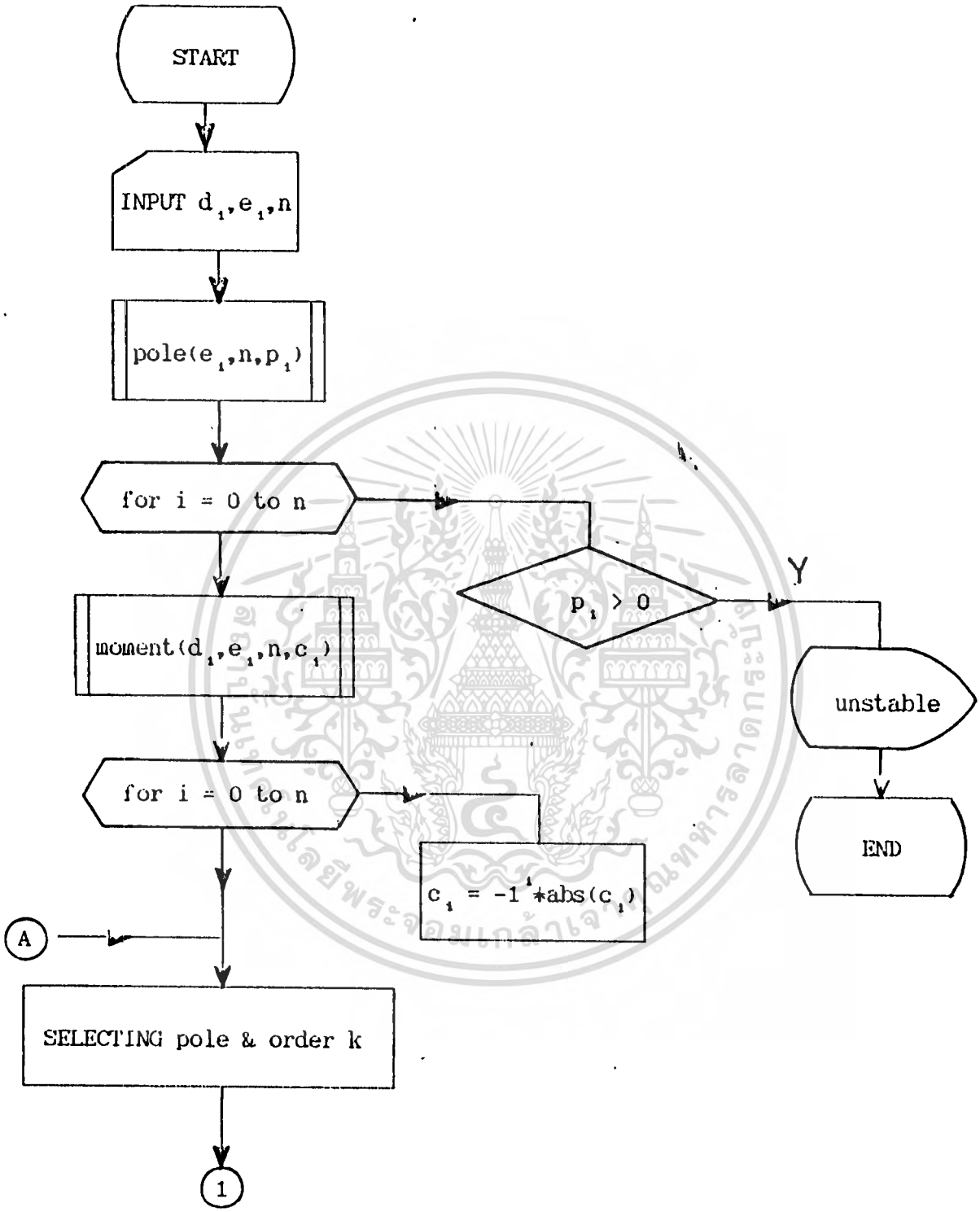
รูปที่ จ.2 (ต่อ) FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมการหาพารามิเตอร์ของ MODEL REDUCTION ใน DOMAIN ของเวลา



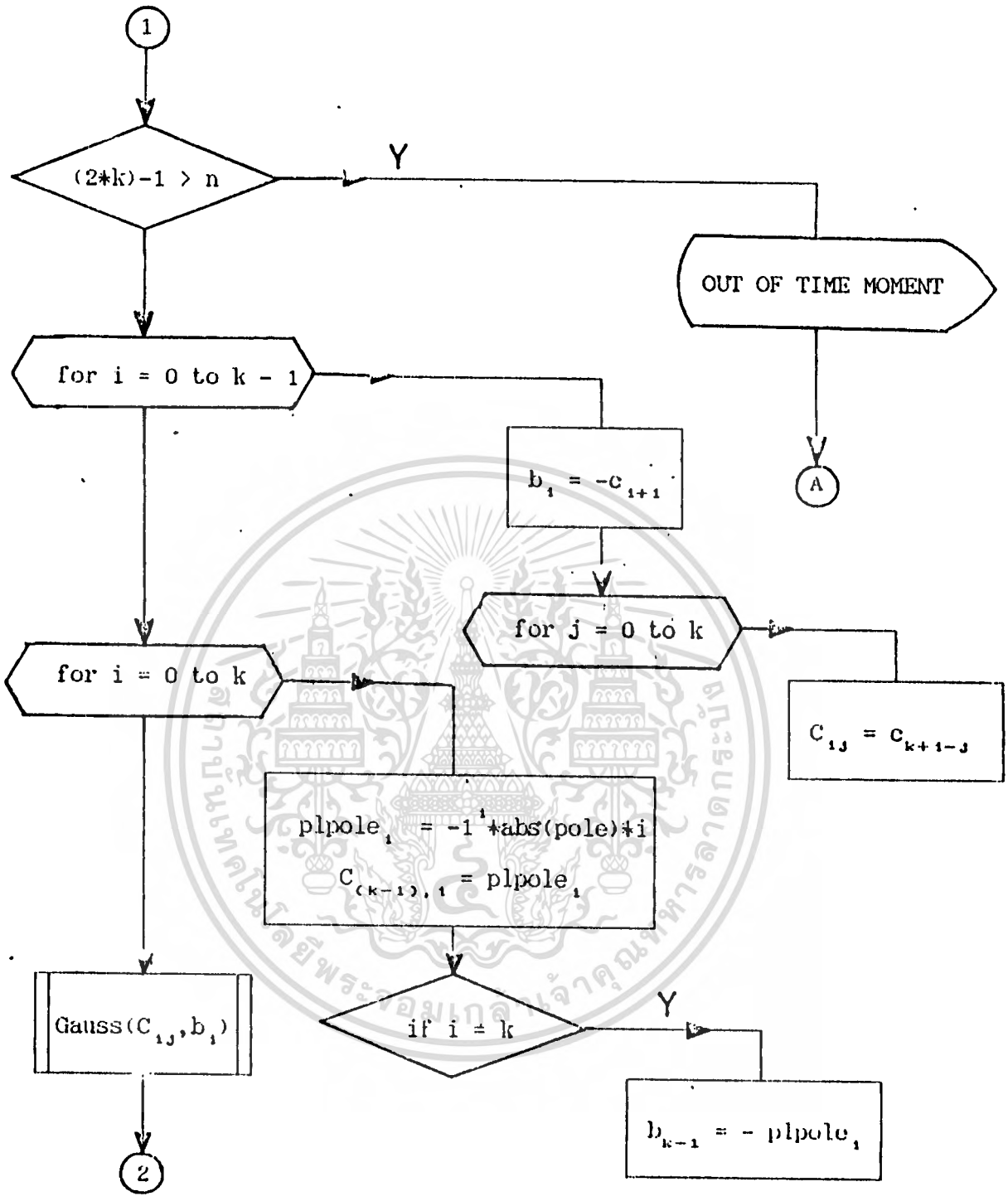
รูปที่ จ.2 (ต่อ) FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมการหาพารามิเตอร์ของ MODEL REDUCTION ใน DOMAIN ของเวลา



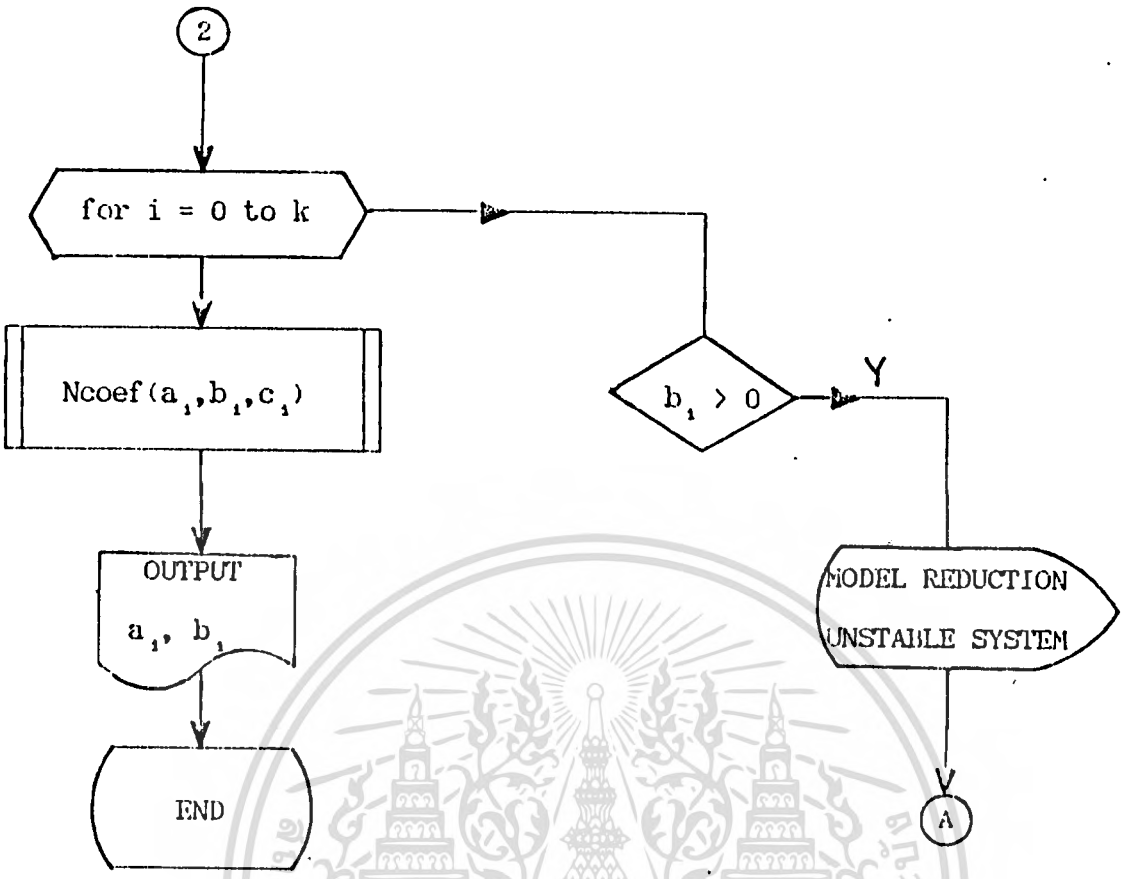
รูปที่ จ.2 (ต่อ) FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมการหาพารามิเตอร์ของ MODEL REDUCTION ใน DOMAIN ของเวลา



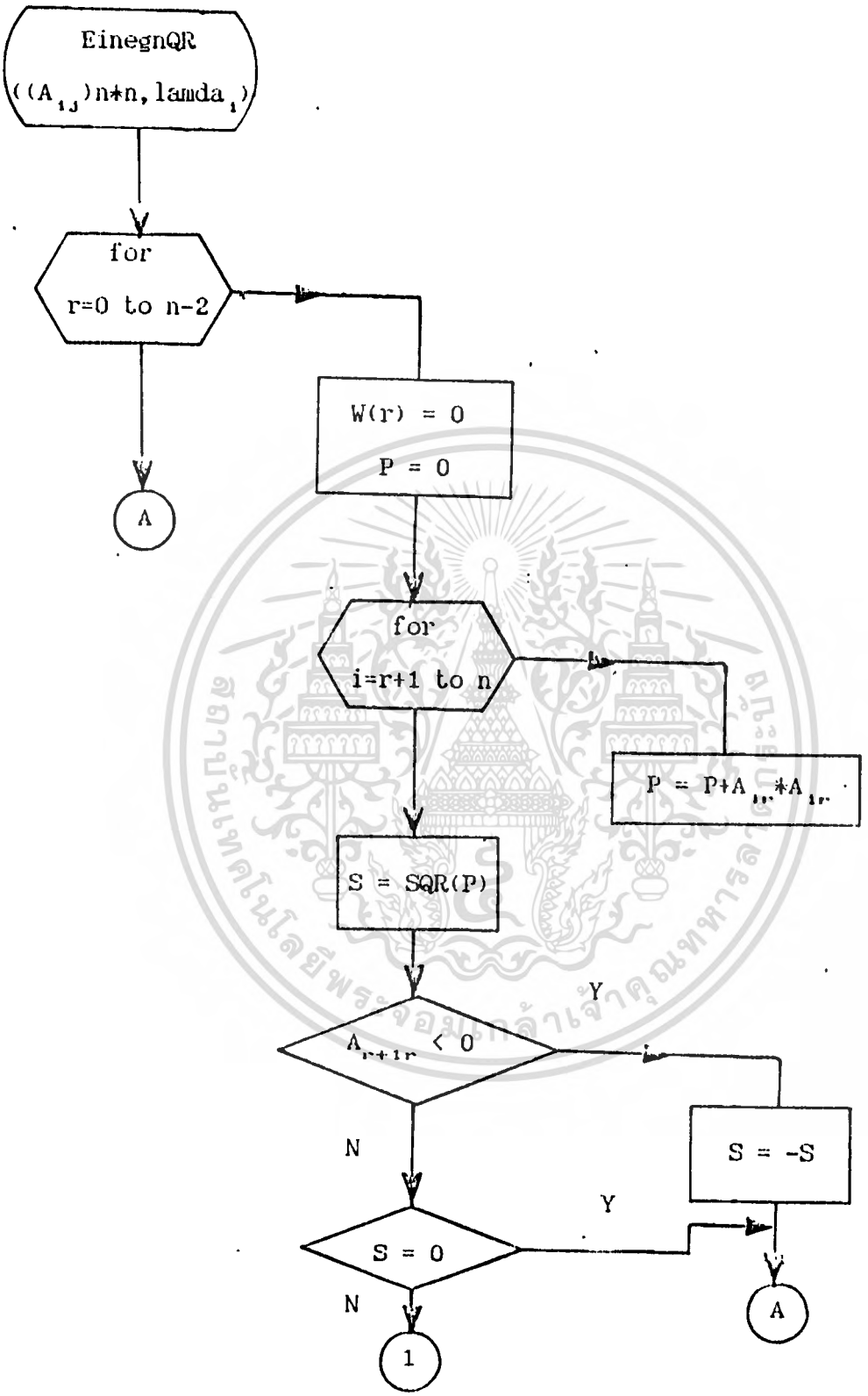
รูปที่ จ.3 FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมการหาสัมประสิทธิ์ของ MODEL REDUCTION ใน DOMAIN ของความถี่



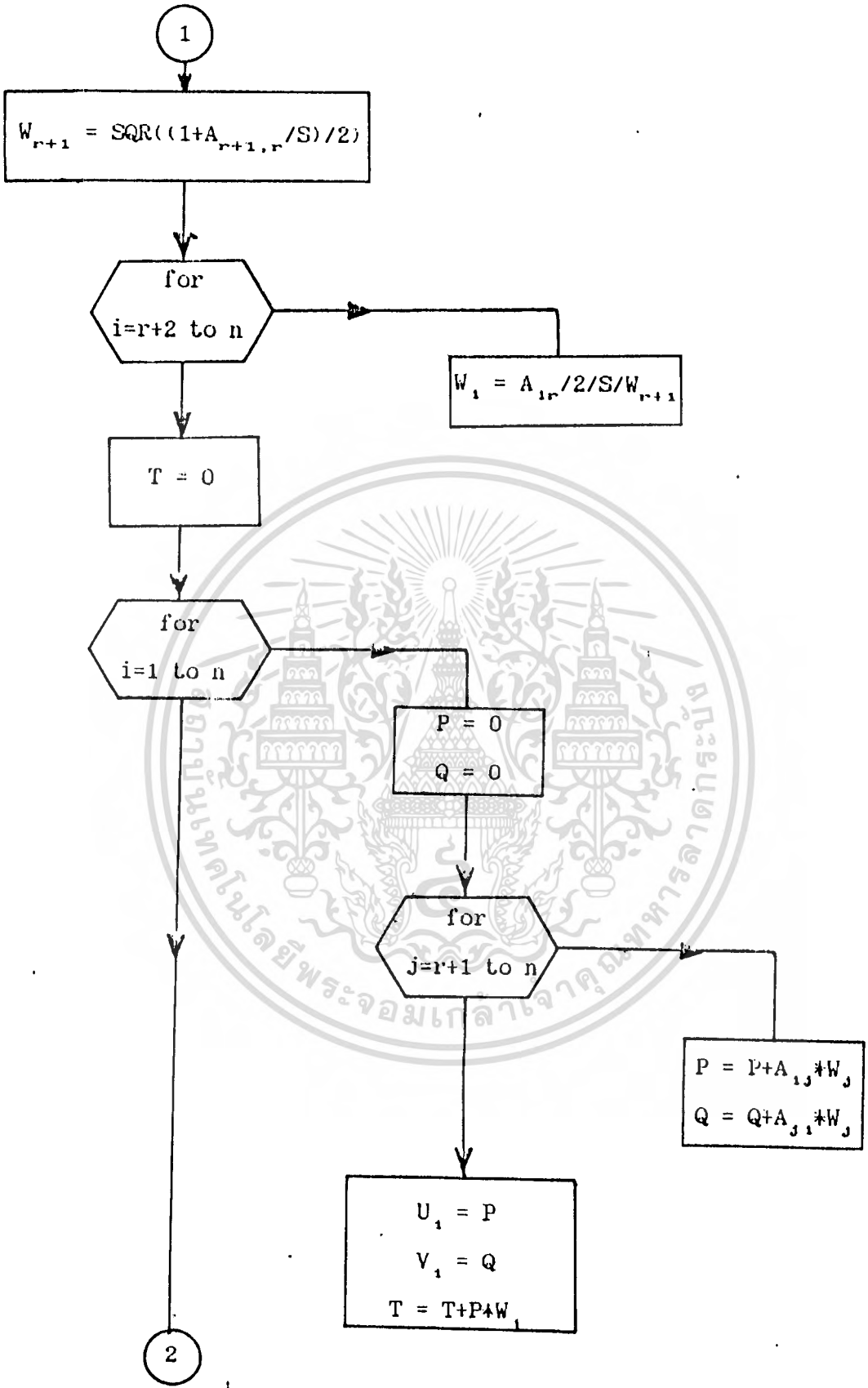
รูปที่ จ.3 FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมการหาสัมประสิทธิ์ของ MODEL REDUCTION ใน DOMAIN ของความถี่



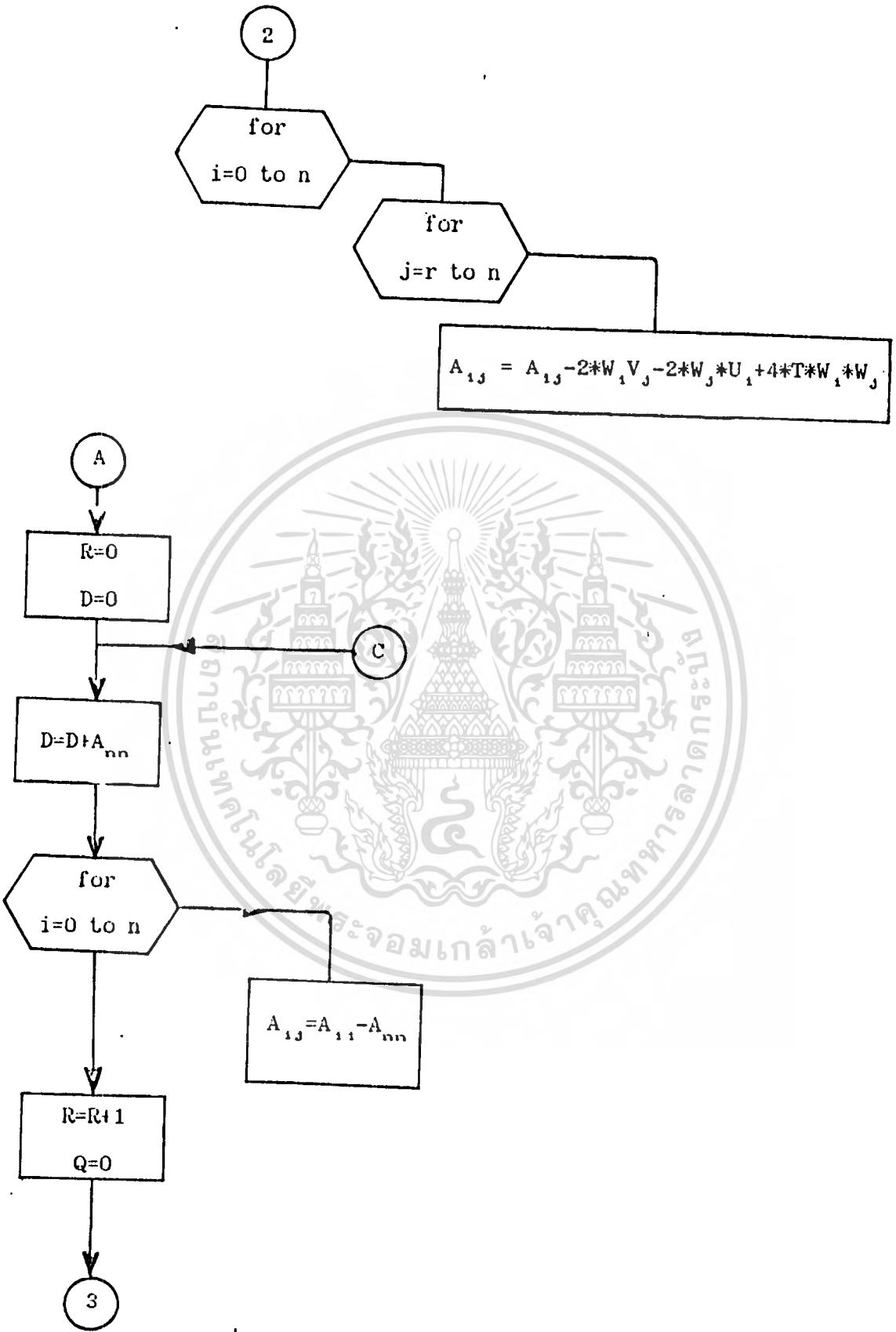
รูปที่ จ.3 (ต่อ) FLOW CHART ที่ใช้ในการพัฒนาโปรแกรมการหาสัมประสิทธิ์ของ MODEL REDUCTION ใน DOMAIN ของความถี่



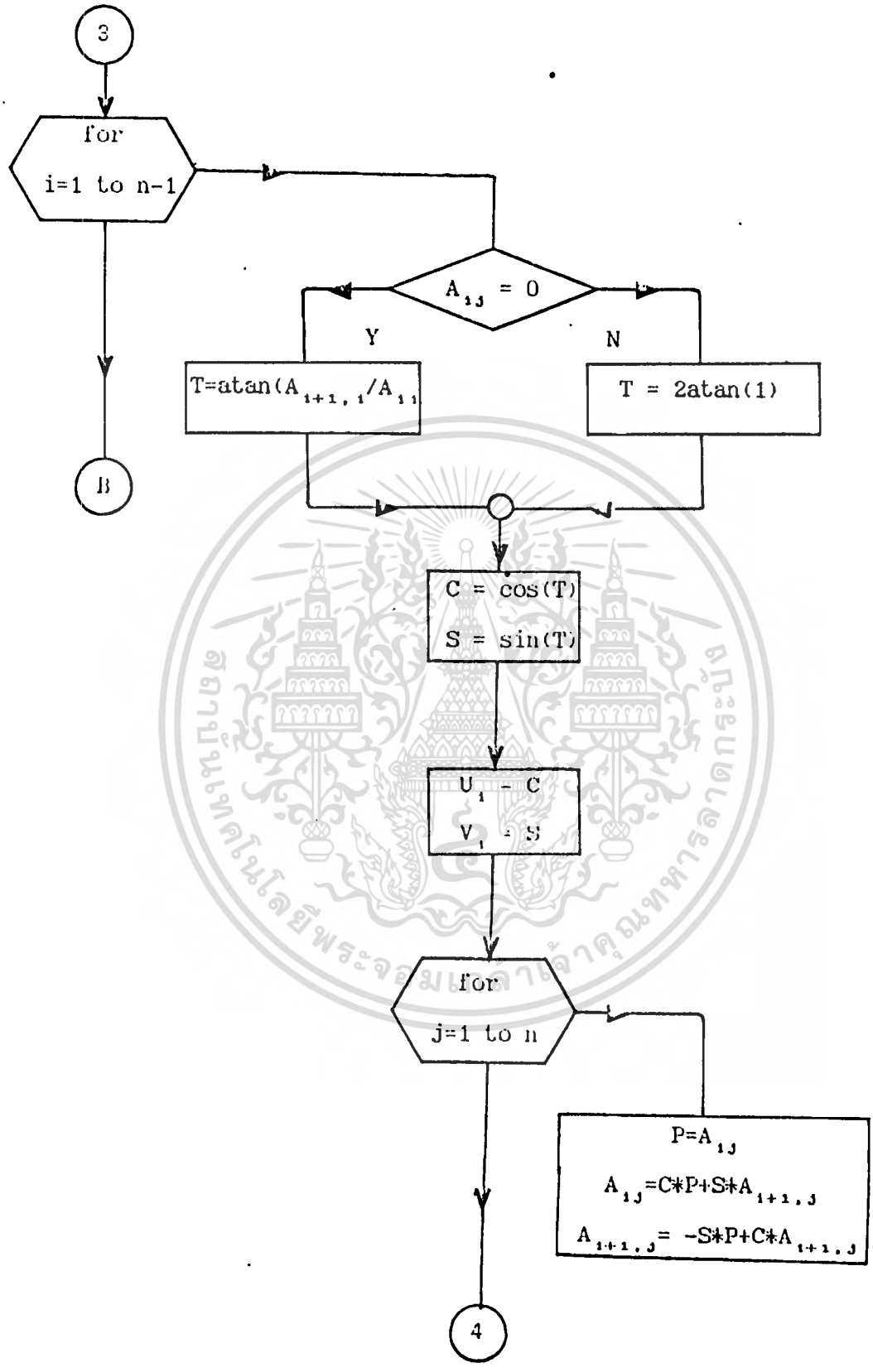
รูปที่ จ.4 FLOW CHART โปรแกรมย่อย EiegnQR



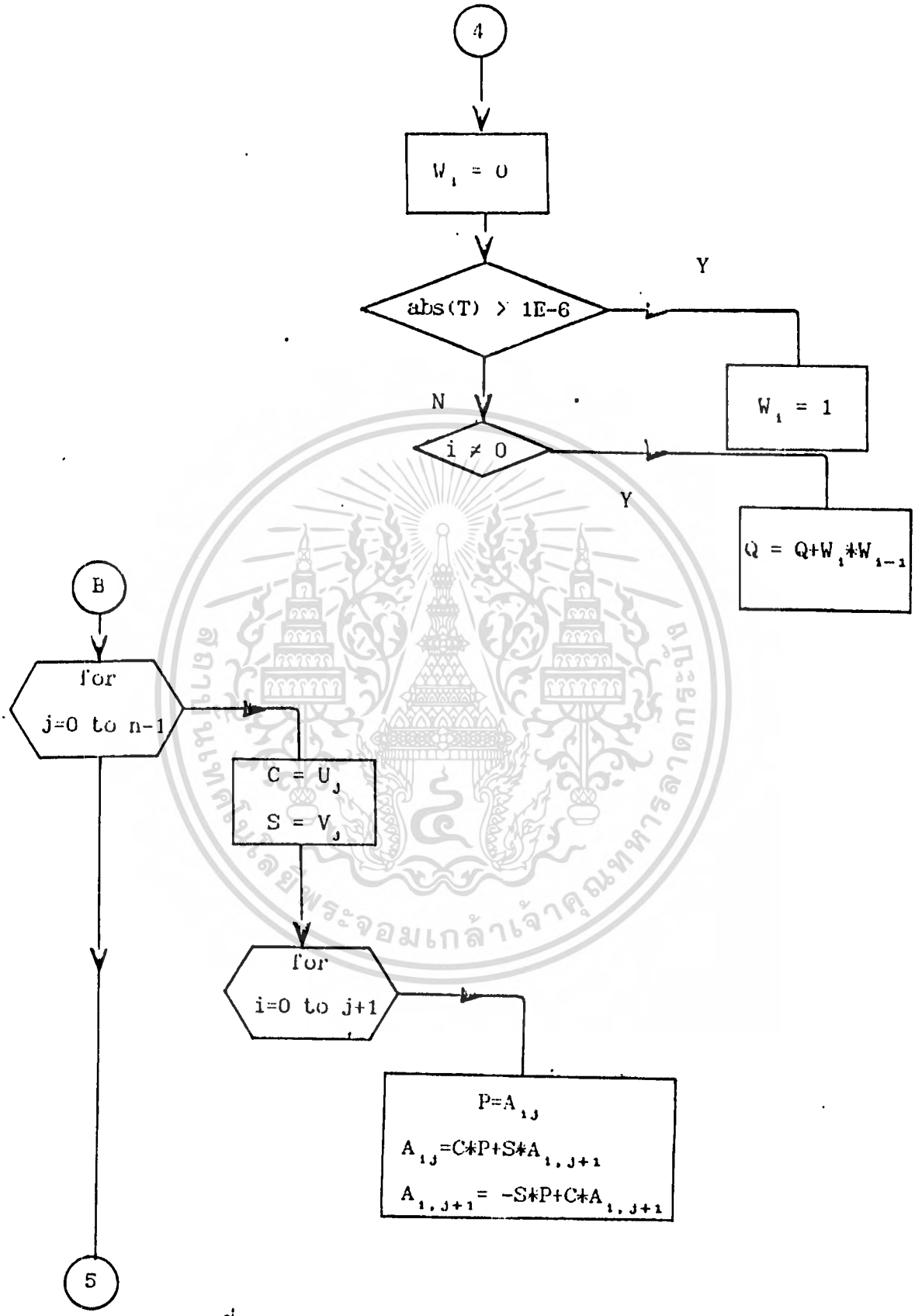
รูปที่ จ.4 (ต่อ) FLOW CHART โปรแกรมย่อย EiegnQR



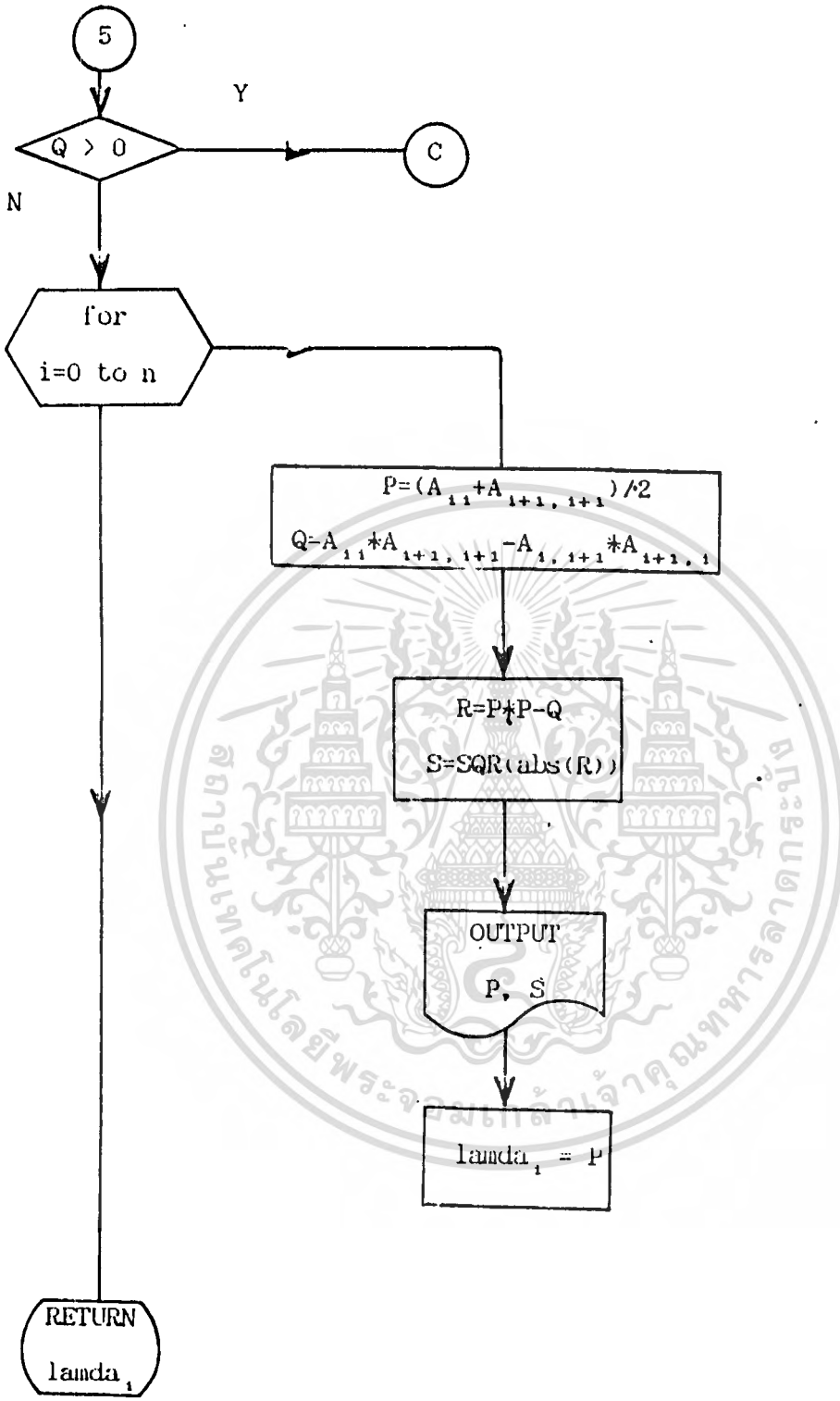
รูปที่ จ.4 (ต่อ) FLOW CHART โปรแกรมย่อย EiegnQR



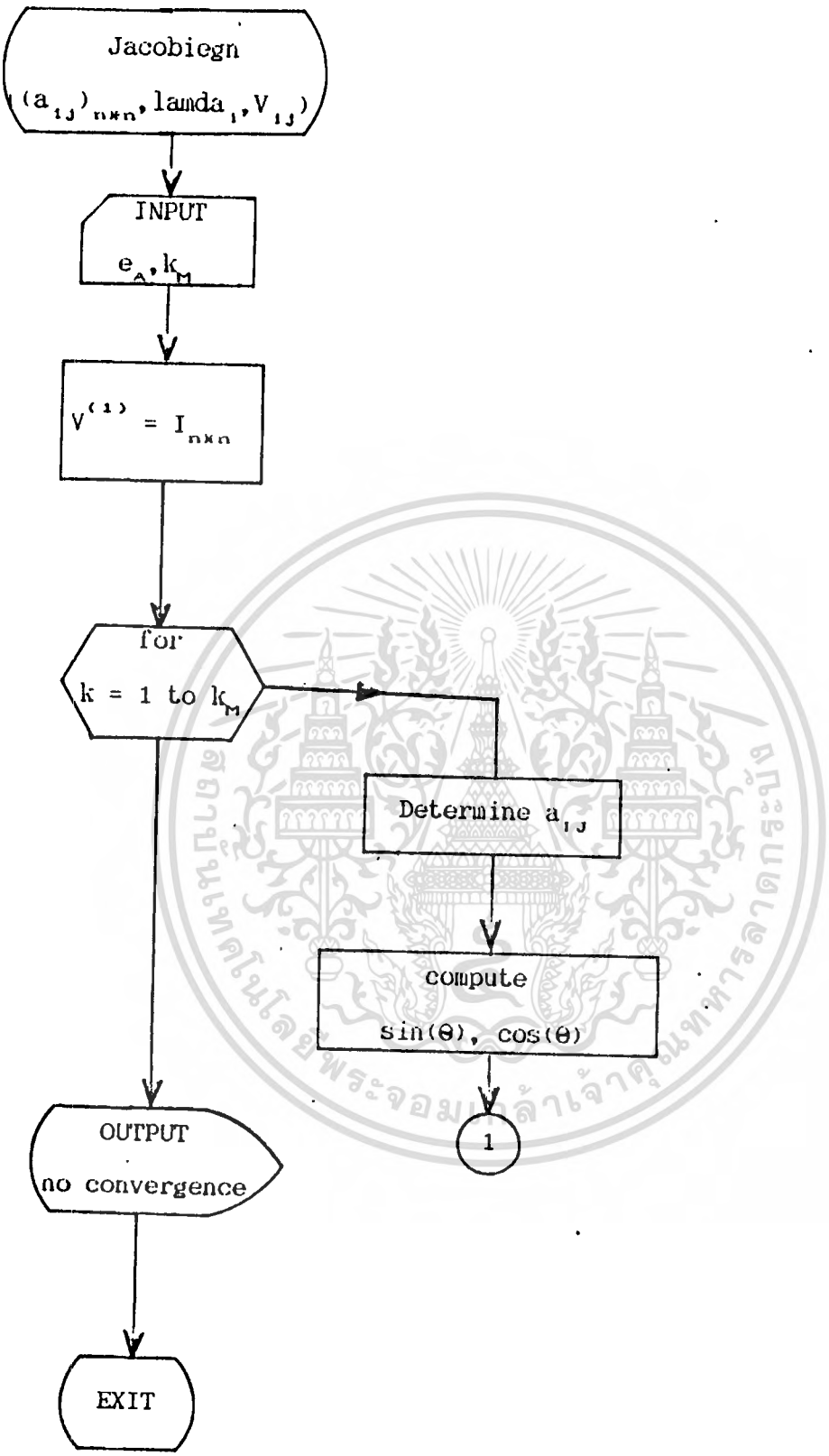
รูปที่ จ.4 (ต่อ) FLOW CHART โปรแกรมย่อย EigenQR



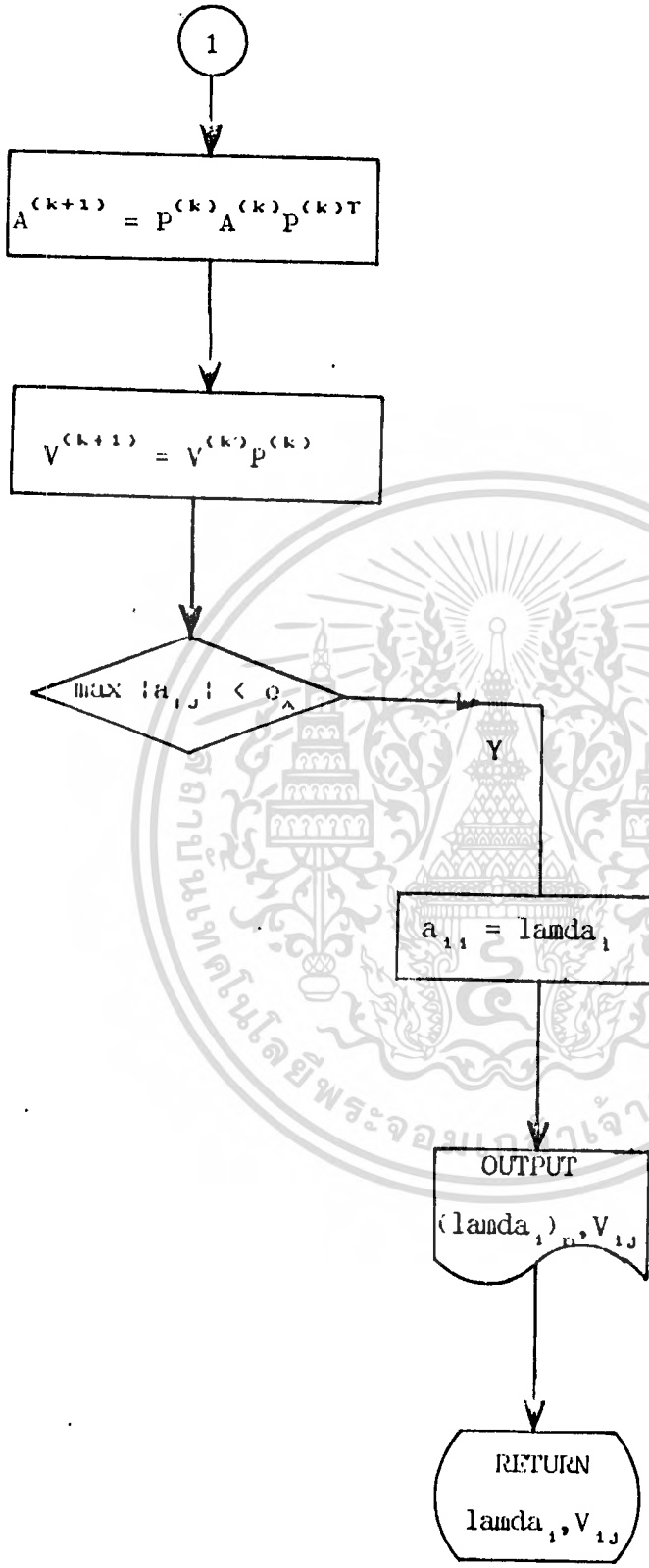
รูปที่ จ.4 (ต่อ) FLOW CHART โปรแกรมย่อย EiegnQR



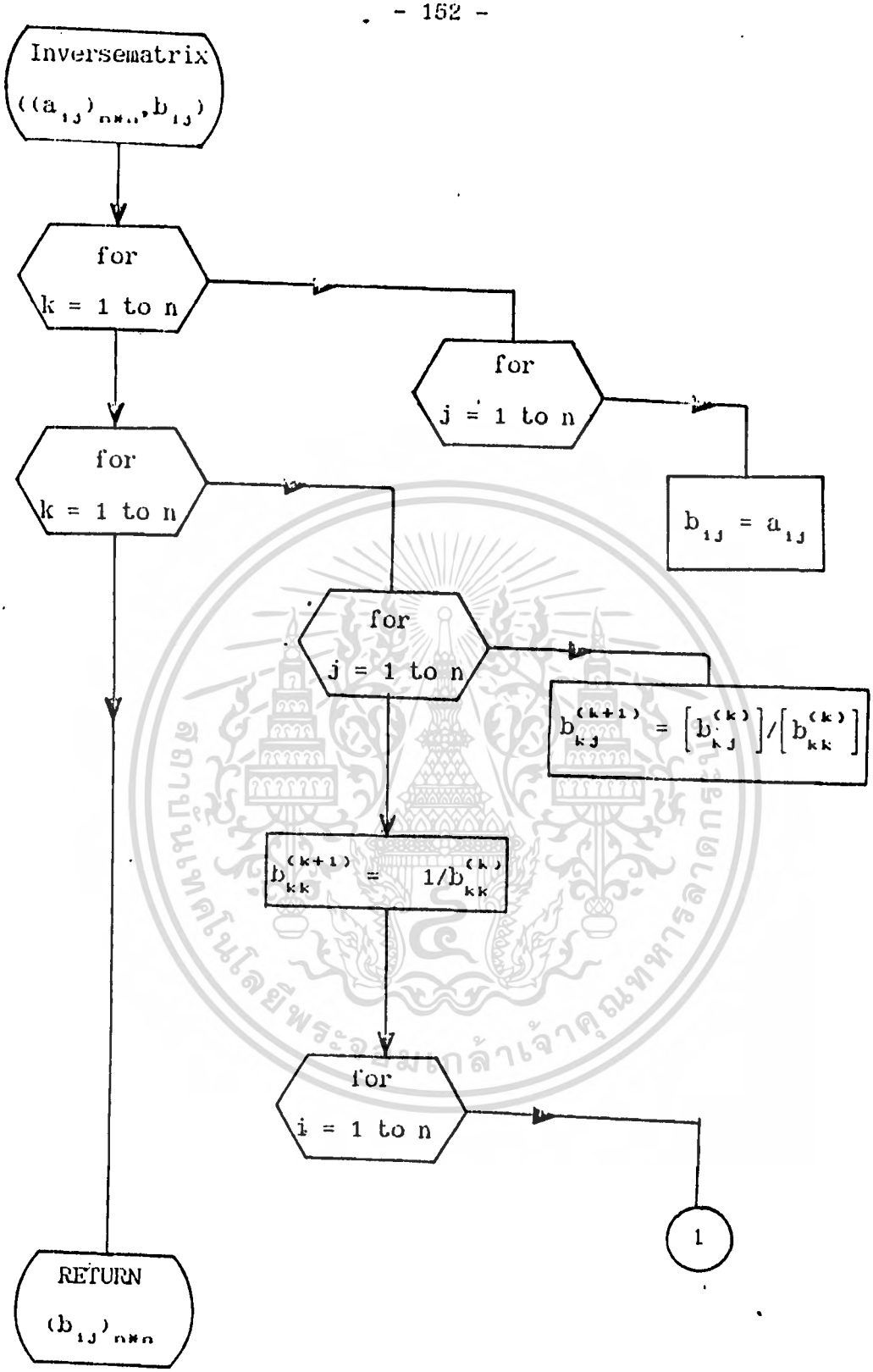
รูปที่ จ.4 (ต่อ) FLOW CHART โปรแกรมย่อย EiegnQR



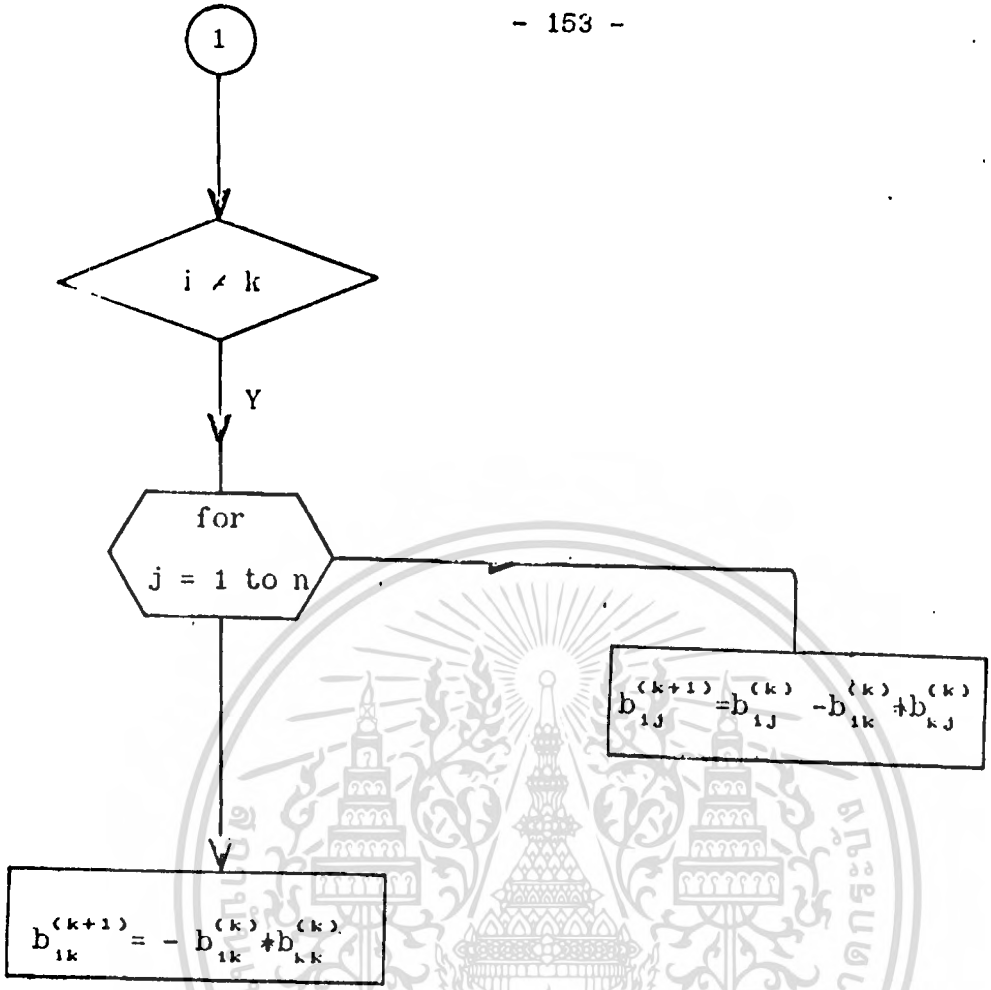
รูปที่ จ.5 FLOW CHART โปรแกรมย่อย Jacobiegn



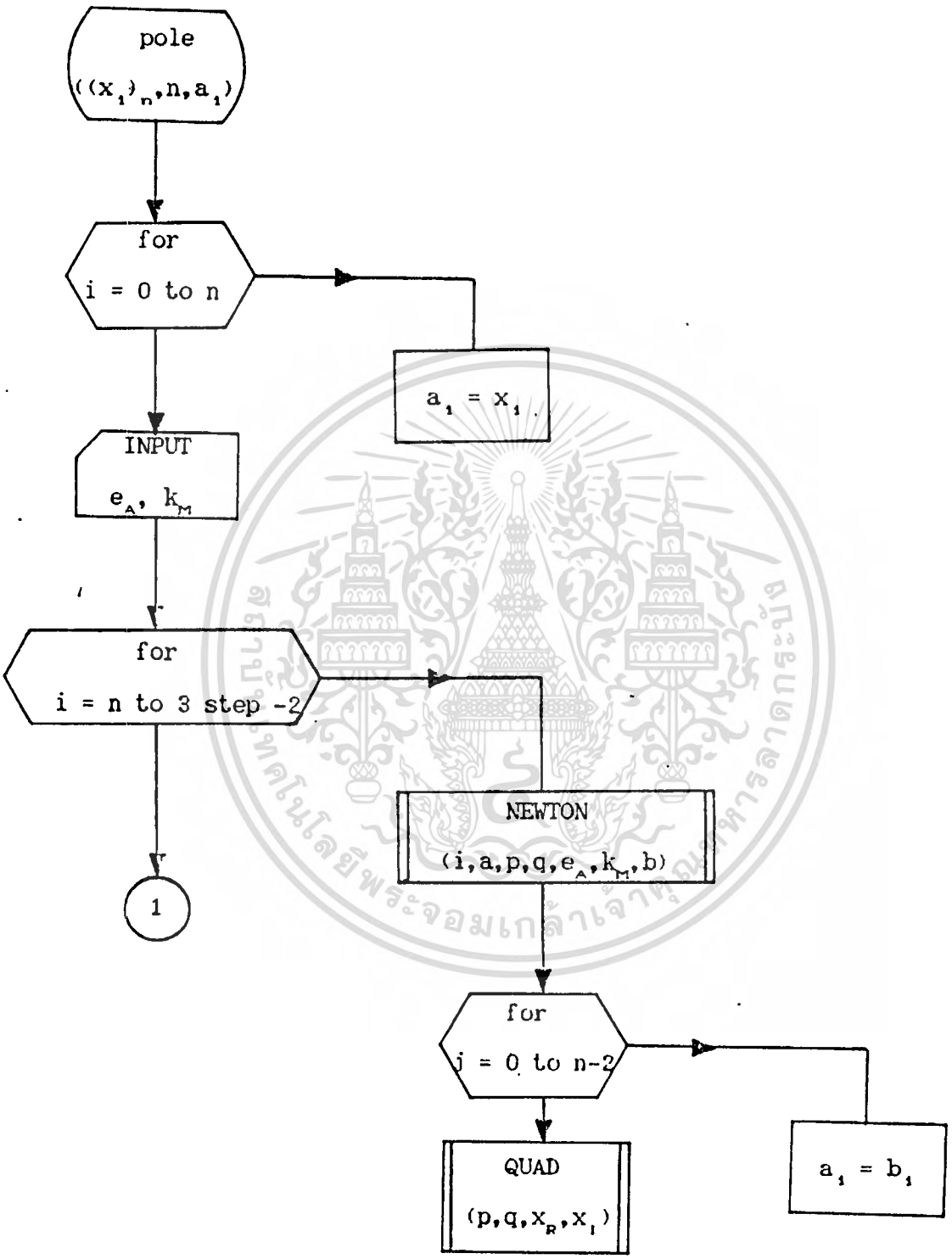
รูปที่ จ.5 (ต่อ)FLOW CHART โปรแกรมย่อย Jacobiegn



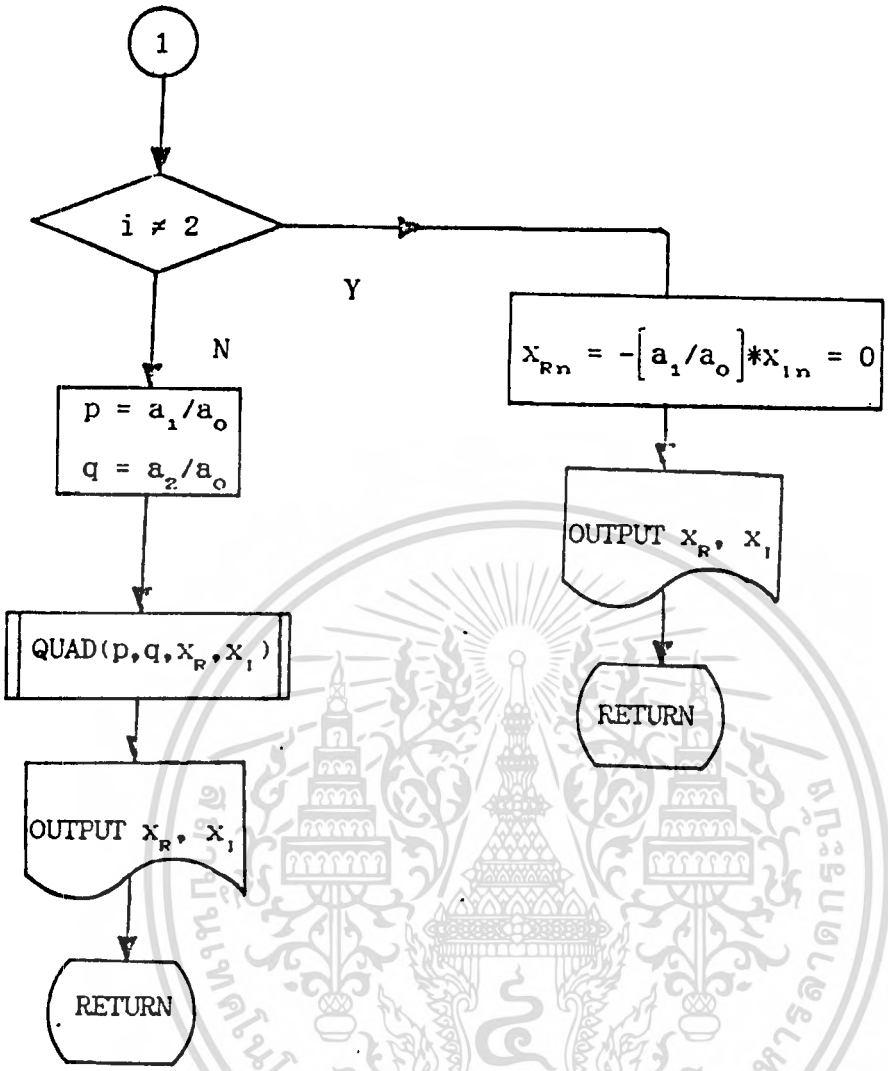
รูปที่ จ.6 FLOW CHART โปรแกรมย่อย Inversematrix



รูปที่ จ.6 (ต่อ) FLOW CHART โปรแกรมย่อย Inversematrix



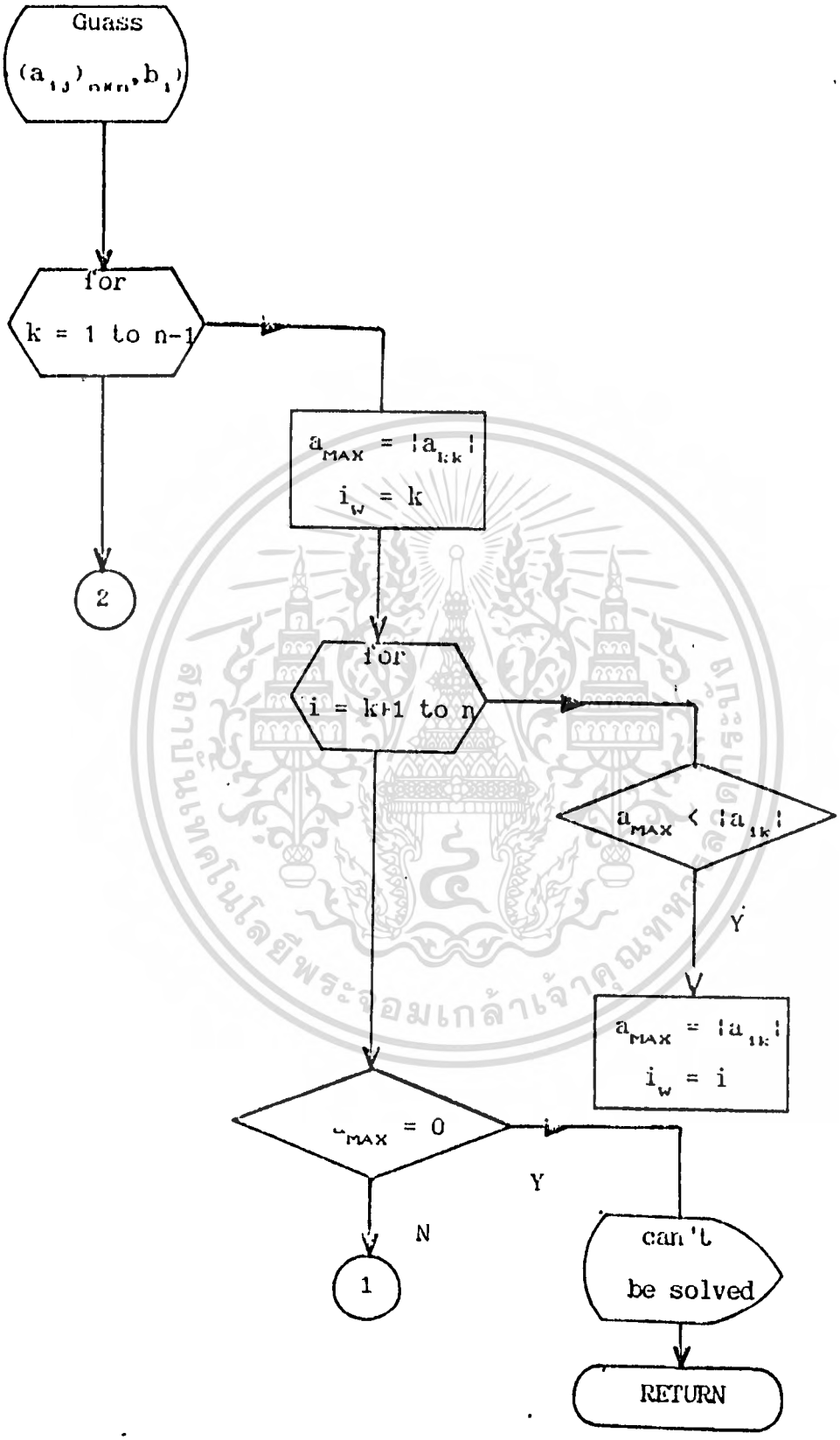
รูปที่ จ.7 FLOW CHART โปรแกรมย่อย pole



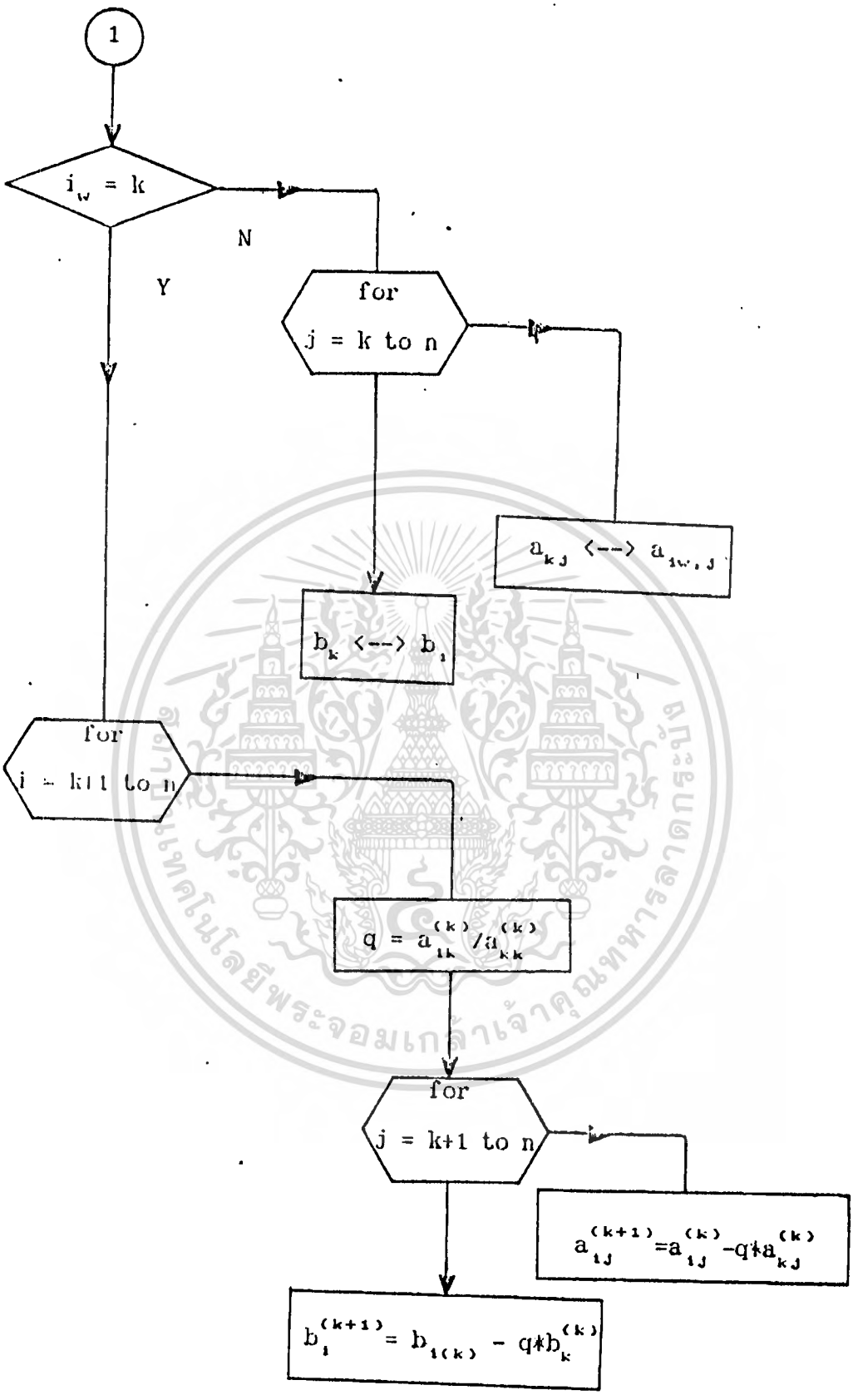
NEWTON : โปรแกรมย่อยของวิธี NEWTON

QUAD : โปรแกรมย่อยสำหรับการหารากสมการกำลังสอง

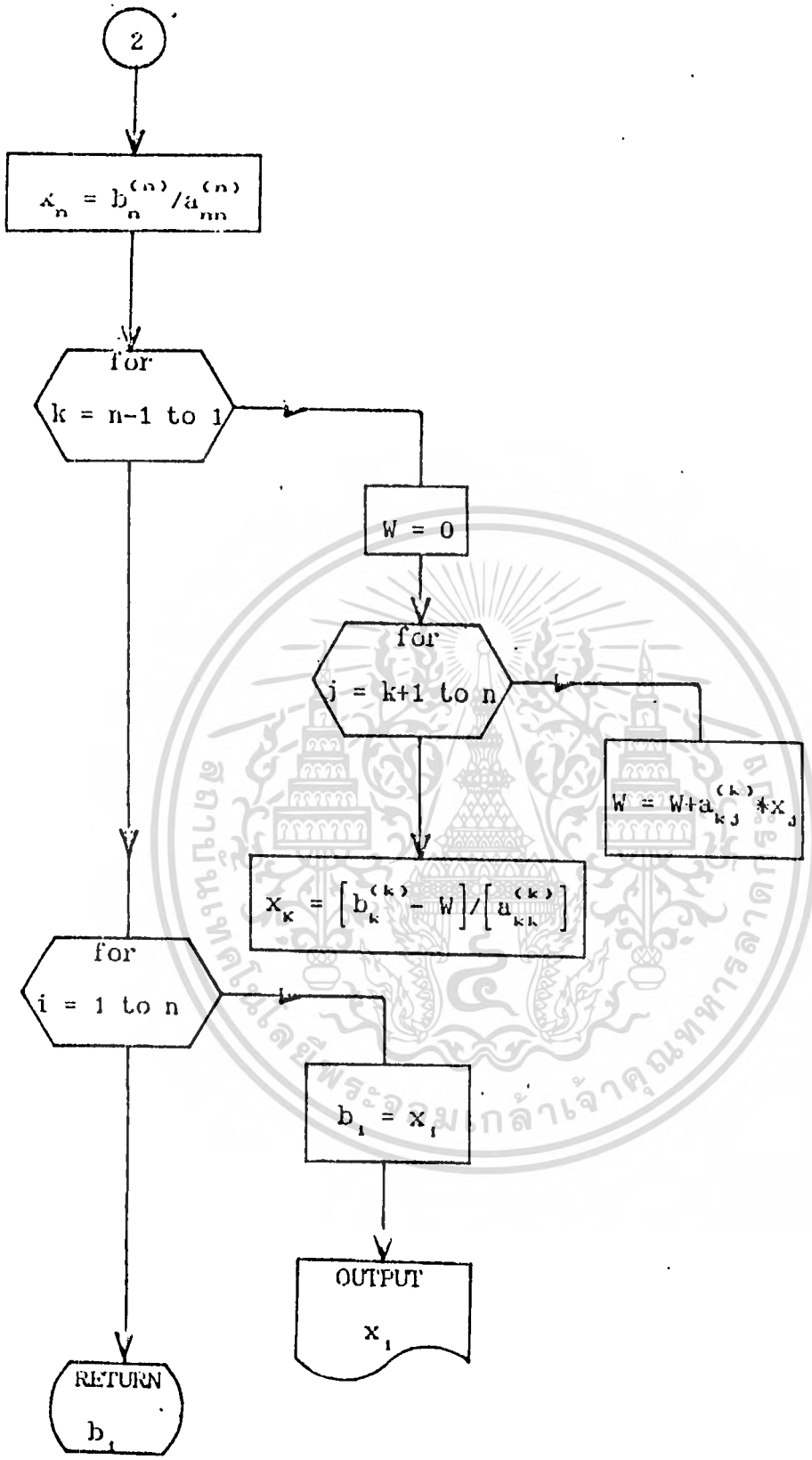
รูปที่ จ.7 (ต่อ) FLOW CHART โปรแกรมย่อย pole



รูปที่ จ.8 FLOW CHART โปรแกรมย่อย Guass



รูปที่ จ.8 (ต่อ) FLOW CHART โปรแกรมย่อย Guass



รูปที่ จ.8 (ต่อ) FLOW CHART โปรแกรมย่อย Guass

ภาคผนวก ฉ.

ผลงานวิจัยที่ได้รับการตีพิมพ์

1. รศ. วิพันธ์ ปรีชาพานิช, สัญญา สายะสถิตย์, " การวิเคราะห์ Reduced Model ด้วยไมโครคอมพิวเตอร์ ", วารสารการประชุมวิชาการวิศวกรรมไฟฟ้า ครั้งที่ 11 สถาบันเทคโนโลยีราชมงคล, 2531, เล่มที่ 1 หน้าที่ 1-22-1 ถึง 1-22-12

2. รศ. วิพันธ์ ปรีชาพานิช, สัญญา สายะสถิตย์, " การวิเคราะห์ Reduced Model ในโดเมนของความถี่ ", วารสาร วิชาการกรมการศึกษาวิจัย, กองบัญชาการทหารสูงสุด, ฉบับที่ 77, 2532, หน้าที่ 43 ถึง 52

3. รศ. วิพันธ์ ปรีชาพานิช, สัญญา สายะสถิตย์, " การลดทอนตัวแบบในระบบเชิงเส้นด้วยไมโครคอมพิวเตอร์ ", วารสารการประชุมวิชาการวิศวกรรมไฟฟ้า ครั้งที่ 13, มหาวิทยาลัยเชียงใหม่, 2533, หน้า 450 ถึง 461