# การประมาณฟังก์ชั่นในโดเมนเวลาโดย วิธีเน็คกาทีฟ-เกรเดียนท์ด้วยการกำหนดโพลเริ่มต้น

# Time Domain Approximation by using Negative Gradient with Prescribed Poles



ธนิตพงษ์ วิบูลยานนท์
Thanitpong Wibuyarnon



อาจารย์ที่ปรึกษา
ผู้ช่วยศาสตราจารย์ ดร. กนก เจนจิระพงศ์เวช
Advisor
Asst.Pro. Dr. Kanok Jainjirapongvej



วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2536

ISBN 974-8158-98-5

หัวข้อวิทยานิพนธ์ การประมาณฟังก์ชั่นในโดเมนเวลาโดยวิธีเน็ตกาทีฟ-เกรเดียนท์ด้วยการกำหนดโพลเริ่มต้น
นักศึกษา นายธนิตพงษ์ วิบูลยานนท์
อาจารย์ที่ปรึกษา ผศ. ดร. กนก เจนจิระพงศ์เวช
ปีการศึกษา 2536

## บทคัดย่อ

วิทยานิพนธ์นี้เสนอ การประมาณสัญญาณพัลซ์ซายน์กำลังสองในโดเมนเวลา โดยวิธีเน็ตกาทีฟ-เกรเดียนท์ ทำการหาจุด Optimum ตามกรรมวิธี Optimization โดยใช้เทคนิคการกำหนดค่าเริ่มต้น ด้วยวิธี ทรานส์ซิทชั่นนัล อุลตร้าสเฟียริคัล-ทอมป์สัน โพลิโนเมียล (Transitional Ultraspherical-Thomson Polynomial) และใช้วิธีการทางคณิตศาสตร์ในการประยุกต์ใช้งานร่วมกับเครื่องไมโครคอมพิวเตอร์ เพื่อคำนวณหาสัมประสิทธิ์ต่างๆ ของทรานส์เฟอร์ฟังก์ชั่น โดยตรวจสอบค่าผิดพลาดในแนวทางลบ (Negative Gradient) ทุกๆครั้งของการคำนวณซ้ำ (Iterative) พร้อมทั้งทำการปรับปรุงค่าสัมประสิทธิของฟังก์ชั่น แล้วนำมาเปรียบเทียบกับฟังก์ชั่นของพัลซ์ซายน์กำลังสองทางอุดมคติ เพื่อให้ค่าผิดพลาดน้อยที่สุด จากทรานส์เฟอร์ฟังก์ชั่นที่ได้นั้นนำไปประกอบวงจรเน็ทเวิร์ก เพื่อสร้างสัญญาณพัลซ์ซายน์กำลังสอง

| | |
|---|---|
| Thesis Title | Time Domain Approximation by using Negative-Gradient with Prescribed Poles |
| Name | Mr.Thanitpong Wibuyarnon |
| Thesis Advisor | Asst.Prof. Dr.Kanok Jainjirapongvej |
| Level of Study | Master of Engineering in Electrical Engineering |
| Academic Year | 1993 |

**Abstract**

The thesis imply, the sine squares pulse is approximated in time domain by using optimization technique. In order to reduce the computational time of iteration, the technique of determination the initial values by using Transitional Ultraspherical - Thomson (TUT) polynomials is proposed. Form these initial values, The Negative-Gradient Method is utilized to continue searching their optimum coefficients of the approximation transfer function. Herein, it is shown that an appropriate choosing of the TUT parameters make it possible to approximate sine squares pulse response with satisfy least error, the approximated transfer function can be realized by network circuit to excite sine squares impulse which has the same response curve as the approximation process.

# กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ดีนั้น ขอขอบพระคุณผู้ช่วยศาสตราจารย์ ดร.กนก เจนจิระพงศ์เวช เป็นอย่างสูง ที่ได้ประสิทธิ์ประสาทวิชาการให้แก่ผู้เขียน ตลอดจนช่วยแนะนำและให้คำปรึกษาในการทำวิทยานิพนธ์ ผู้เขียนขอกราบขอบพระคุณเป็นอย่างสูง และขอขอบคุณท่านอาจารย์ เจ้าหน้าที่ภาควิชาเทคนิคอุตสาหกรรมทุกท่าน ที่ได้สนับสนุนในด้านเครื่องมือทดลองจนวิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดีโดยตลอด

# สารบัญ

# บทที่ 1
# บทนำ

การออกแบบวงจรเน็ทเวิร์ค เพื่อให้ได้สัญญาณพัลซ์ชายน์กำลังสอง [11] ซึ่งมีสมการเป็น $\sin^2(\pi t)/2T$ เนื่องจากพัลซ์ชายน์กำลังสองดังกล่าวนิยมใช้เป็นสัญญาณทดสอบ (test signal) ในการส่งสัญญาณภาพในระบบโทรทัศน์ การประมาณเน็ทเวิร์คฟังก์ชั่นเพื่อให้ได้ผลตอบสนองต่อสัญญาณพัลซ์ชายน์กำลังสอง ด้วยกรรมวิธี Optimization นั้น ถ้ากำหนดค่าเริ่มต้นให้กับกรรมวิธี Optimization ที่ไม่เหมาะสม อาจทำให้ต้องใช้เวลานานมากหรืออาจทำให้กรรมวิธี Optimization ล้มเหลวไป เพื่อหลีกเลี่ยงผลดังกล่าวข้างต้น ในวิทยานิพนธ์นี้จึงได้เสนอการประมาณฟังก์ชั่นในโดเมนเวลา ด้วยวิธีเน็คกาทีฟ-เกรเดียนท์ (Negative Gradient) [17,20] โดยการกำหนดโพลเริ่มต้นด้วยวิธี ทรานส์ซิทชั่นนัล อุลตร้าสเฟียริคัล-ทอมป์สันโพลิโนเมียล (Transitional Ultraspherical-Thomson polynomials [TUT]) [8] เพื่อให้ได้ผลตอบสนองของพัลซ์ชายน์กำลังสองที่ดีที่สุดระหว่างทรานส์ซิทชั่นนัล อุลตร้าสเฟียริคัลโพลิโนเมียล กับทรานส์ซิทชั่นนัลทอมป์สันโพลิโนเมียล [2,15] ดังนั้นการกำหนดค่าเริ่มต้นที่ใกล้จุด Optimum จะทำให้การนำเอาวิธีเน็คกาทีฟ-เกรเดียนท์มาใช้เหมาะในการประมาณทรานส์เฟอร์ฟังก์ชั่น และทำให้การค้นหาจุด Optimum อยู่ในแนวทางที่ถูกต้องและรวดเร็วขึ้น

## 1.1 วัตถุประสงค์ของวิทยานิพนธ์

1. เพื่อศึกษาวิธีการทางคณิตศาสตร์ และนำมาประยุกต์ใช้งานทางวิศวกรรม
2. เพื่อศึกษาแนวทางการประมาณฟังก์ชั่นในโดเมนเวลา
3. เพื่อศึกษาเทคนิคในกรรมวิธี Optimization
4. เพื่อศึกษาแนวทางการออกแบบวงจรเน็ทเวิร์กฟังก์ชั่นอย่างมีประสิทธิภาพ

## 1.2 เนื้อหาของวิทยานิพนธ์

สำหรับวิทยานิพนธ์นี้เป็นการประมาณฟังก์ชั่นพัลซ์ชายน์กำลังสองในโดเมนของเวลา โดยการกำหนดโพลเริ่มต้นด้วยวิธี ทรานส์ซิทชั่นนัล อุลตร้าสเฟียริคัล-ทอมป์สันโพลิโนเมียล (TUT)ซึ่งได้มาจากวงจรกรองความถี่ต่ำแบบอุลตร้าสเฟียริคัล โพลีโนเมียล และวงจรกรองความถี่ต่ำแบบทอมป์สัน โพลีโนเมียล จากทรานส์เฟอร์ฟังก์ชั่น TUT ที่ได้จึงนำไปประมาณฟังก์ชั่นพัลซ์ชายน์กำลังสองในโดเมนเวลา เปรียบเทียบสัญญาณพัลซ์ที่ประมาณได้กับสัญญาณพัลซ์ทางอุดมคติ คำนวณค่าผิดพลาดของสัญญาณที่ประมาณ และหาทรานส์เฟอร์ฟังก์ชั่น TUT ที่ให้ค่าผิดพลาดจากการประมาณสัญญาณพัลซ์ชายน์กำลังสองทางอุดมคติน้อยที่สุด หลังจากนั้นจึงนำทรานส์เฟอร์ฟังก์ชั่น TUT ที่ได้นำไป

กำหนดเป็นค่าสัมประสิทธิ์เริ่มต้นในการทำกรรมวิธี Optimization ด้วยวิธีเน็คกาทีฟ-เกรเดียนท์ ตรวจสอบค่าผิดพลาดทุกๆครั้งของการคำนวณซ้ำ (Iterative) แต่ละครั้งแล้วนำมาเปรียบเทียบกับฟังก์ชั่นพัลซ์ชายน์กำลังสองทางอุดมคติ เพื่อให้ค่าผิดพลาดน้อยที่สุด (Optimum) ในท้ายสุดนำทรานส์เฟอร์ฟังก์ชั่นที่จุด Optimum นำไปออกแบบวงจรเน็ทเวิกร์ฟังก์ชั่น เพื่อสร้างสัญญาณพัลซ์ชายน์กำลังสองเพื่อใช้งานจริง

ในบทที่ 2 กล่าวถึงผลตอบสนองของเน็ทเวิกร์ฟังก์ชั่น ทฤษฎีพัลซ์ชายน์กำลังสองที่ใช้ในการตรวจสอบข้อบกพร่องของระบบการส่งสัญญาณโทรทัศน์ และคุณสมบัติของเกรเดียนท์-เวคเตอร์ที่ใช้ในกรรมวิธี Optimization

ในบทที่ 3 กล่าวถึงวงจรกรองความถี่ต่ำแบบอุลตร้าสเฟียริคัลโพลีโนเมียล วงจรกรองความถี่ต่ำแบบทอมป์สัน โพลีโนเมียล วงจรกรองความถี่ต่ำแบบบัตเตอร์เวิร์ท โพลิโนเมียล การออกแบบทรานส์ซิทชั่นนัล อุลตร้าสเฟียริคัล-ทอมป์สันโพลิโนเมียล (TUT) การออกแบบทรานส์ซิทชั่นนัล บัตเตอร์เวิร์ท-ทอมป์สันโพลิโนเมียล (TBT) การประมาณฟังก์ชั่นพัลซ์ชายน์กำลังสองด้วยทรานส์ซิทชั่นนัล อุลตร้าสเฟียริคัล-ทอมป์สันโพลิโนเมียล และการประมาณฟังก์ชั่นพัลซ์ชายน์กำลังสองด้วยทรานส์ซิทชั่นนัล บัตเตอร์เวิร์ท-ทอมป์สันโพลิโนเมียล จากนั้นคำนวณหาค่าผิดพลาดจากการประมาณพัลซ์ชายน์กำลังสอง และผลตอบสนองของฟังก์ชั่นที่ประมาณ

ในบทที่ 4 กล่าวถึงกรรมวิธี Optimization ด้วยวิธี เน็คกาทีฟ-เกรเดียนท์ จากการกำหนดค่าสัมประสิทธิเริ่มต้นด้วยทรานส์เฟอร์ฟังก์ชั่น TUT จากนั้นคำนวณหาค่าผิดพลาดจากการประมาณพัลซ์ชายน์กำลังสอง และผลตอบสนองของฟังก์ชั่นที่ประมาณ

ในบทที่ 5 เป็นการออกแบบวงจรแอ็คทีฟเน็ทเวิกร์จากทรานส์เฟอร์ฟังก์ชั่นที่ได้ การสเกล (Scaling) ค่าอุปกรณ์ทางขนาด (Magnitude Scaling) การสเกลค่าอุปกรณ์ทางความถี่ (Frequency Scaling) และในท้ายสุดเป็นผลตอบสนองจากการสร้างสัญญาณพัลซ์ชายน์กำลังสองเพื่อใช้งานจริง

ในบทที่ 6 ซึ่งเป็นบทท้ายสุดจะสรุปผลการวิจัย และข้อเสนอแนะเพื่อเป็นแนวทางในการพัฒนาให้มีประสิทธิภาพต่อไป

ภาคผนวก เป็นรายละเอียดของโปรแกรมทั้งหมดในการคำนวณหาค่าผลลัพธ์ต่างๆ ซึ่งสามารถแสดงผลได้ 3 ทาง คือ 1.จอแสดงผล (Monitor) 2.เครื่องพิมพ์ (Printer) 3.เอกซ์-วายพล๊อทเตอร์ (X-Y Plotter)

1.3 ประโยชน์ที่ได้รับจากวิทยานิพนธ์

1. สามารถนำไปใช้ในการประมาณฟังก์ชั่นในขอบข่ายของเวลา (Time Domain Approximation) และให้ได้ผลตอบสนองต่อสัญญาณอิมพัลซ์ใกล้เคียงกับสัญญาณที่ต้องการในทางอุดมคติ

2. สามารถนำไปใช้ในการประมาณฟังก์ชั่นในขอบข่ายความถี่ (Frequency Domain Approximation) เพื่อให้ได้คุณลักษณะทางขนาดหรือกรุ๊ฟดีเลย์ใกล้เคียงกับคุณลักษณะทางขนาดหรือกรุ๊ฟดีเลย์ตามต้องการ

3. การประมาณฟังก์ชั่นโดยกรรมวิธี Optimization สามารถที่จะวิเคราะห์ ตรวจสอบค่าผิดพลาดของผลตอบสนองต่อสัญญาณอิมพัลซ์จนเป็นที่พอใจ แล้วจึงทำการสร้างวงจรเน็ทเวิกร์เพื่อใช้งานจริง

4. หลักการและการคำนวณหาค่าผลลัพธ์ต่างๆ ในวิทยานิพนธ์นี้ได้เขียนโปรแกรมเพื่อจำลองผลตอบสนอง ซึ่งสามารถนำไปเป็นแนวทางในการพัฒนาใช้งานให้มีประสิทธิภาพมากที่สุด

# บทที่ 2
# ทฤษฎีทั่วไป

## 2.1 เน็ทเวิกร์ฟังก์ชั่น

การออกแบบวงจรใด ๆ เพื่อให้ผลตอบสนองตามที่ต้องการ จะทำได้โดยหาเน็ทเวิกร์ฟังก์ชั่นของวงจรเทียบกับฟังก์ชั่นที่ต้องการในทางอุดมคติ ซึ่งค่าที่ได้จะต้องให้ใกล้เคียงกับฟังก์ชั่นในทางอุดมคติมากที่สุด เช่น การออกแบบวงจรกรองความถี่จะต้องให้เน็ทเวิกร์ฟังก์ชั่นที่ได้ยอมให้สัญญาณความถี่ในย่านผ่านสัญญาณ (Passband) ผ่านออกไปได้ทั้งหมดโดยไม่มีการสูญเสียหรือผิดเพี้ยน และสามารถกำจัดสัญญาณความถี่ในย่านหยุดสัญญาณ (Stopband) ได้ทั้งหมดโดยสิ้นเชิง แต่เนื่องจากวงจรใด ๆ สามารถแทนได้แต่เฉพาะฟังก์ชั่นที่อยู่ในรูปโพลีโนเมียลเศษส่วนเท่านั้น จึงจำเป็นต้องทำความเข้าใจเกี่ยวกับผลตอบสนองชนิดต่าง ๆ ของฟังก์ชั่นแบบนี้ ซึ่งผลตอบสนองของเน็ทเวิกร์ฟังก์ชั่นต่อความถี่ [4,5,6] ที่สำคัญ ได้แก่

- ผลตอบสนองทางขนาด (Magnitude Response)
- ผลตอบสนองทางเฟส (Phase Response)

ผลตอบสนองทั้งสองชนิดนี้จะถูกพิจารณาเปรียบเทียบกับความถี่ของสัญญาณอินพุทที่มีการเปลี่ยนแปลง เน็ทเวิกร์โดยทั่วไปสามารถเขียนเป็นทรานส์เฟอร์ฟังก์ชั่นที่อยู่ในรูปโพลิโนเมียลคือ

$$T(s) = \frac{N(s)}{D(s)} = \frac{a_m s^m + a_{m-1} s^{m-1} + \ldots + a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} + \ldots + b_1 s + b_0} \qquad \ldots(2.1)$$

โดยที่ N(s) และ D(s) เป็นโพลิโนเมียลของตัวเศษและโพลิโนเมียลของตัวส่วนตามลำดับ
m เป็นอันดับของโพลิโนเมียลเศษ และ n เป็นอันดับของโพลิโนเมียลส่วน โดยที่ $n \geq m$
a และ b เป็นสัมประสิทธิ์ที่เป็นตัวเลขจำนวนจริง
s เป็นตัวแปรความถี่เชิงซ้อน (Complex-Frequency Variable)

สมการ (2.1) ที่อยู่ในรูปโพลิโนเมียลสามารถแยกตัวประกอบซึ่งจะได้

$$T(s) = K \frac{(s-z_1)(s-z_2)\ldots(s-z_m)}{(s-p_1)(s-p_2)\ldots(s-p_n)} \qquad \ldots(2.2)$$

$$T(s) = K \frac{\prod_{i=1}^{m} (s-z_i)}{\prod_{j=1}^{n} (s-p_j)}$$

โดยที่ K เป็นอัตราขยาย ซึ่งมีค่าคงที่

$z_i$ เป็นตำแหน่งซีโร่อันดับที่ i

$p_j$ เป็นตำแหน่งโพลอันดับที่ j

เมื่อพิจารณาผลตอบสนองทางความถี่ของเน็ทเวิร์ก โดยสมมติป้อนสัญญาณอินพุทเป็นรูปซายน์และเน็ทเวิร์กเป็นแบบลิเนียร์ ผลตอบสนองของวงจรในสภาวะเสถียรของสัญญาณรูปซายน์ ซึ่ง s มีค่าเท่ากับ jω ดังนั้นสมการ (2.2) จะได้

$$T(j\omega) = K \frac{(j\omega-z_1)(j\omega-z_i)\ldots(j\omega-z_m)}{(j\omega-p_1)(j\omega-p_j)\ldots(j\omega-p_n)} \qquad \ldots(2.3)$$

เมื่อพิจารณาแฟคเตอร์ $(j\omega-s_i)$ โดยให้ $s_i$ เป็นซีโร่ คือ $s_i = z_i$ หรือเป็นโพล คือ $s_i = p_i$ ซึ่งค่า $s_i$ จะเป็นตัวเลขเชิงซ้อนสมมติให้

$$s_i = \alpha_i + j\beta_i$$

ค่าแฟคเตอร์ $(j\omega - s_i)$ จะได้เป็น

$$-\alpha_i + j(\omega - \beta_i) = M_i e^{j\theta_i} \qquad \ldots(2.4)$$

เมื่อ $M_i = \sqrt{\alpha_i^2 + (\omega - \beta_i)^2}$

$$\theta_i = \tan^{-1} \frac{\omega - \beta_i}{-\alpha_i}$$

ฉะนั้นสมการ (2.3) สามารถเขียนใหม่เป็น

$$T(j\omega) = K \frac{M_{z1} e^{j\theta_{z1}} M_{zi} e^{j\theta_{zi}} \ldots M_{zm} e^{j\theta_{zm}}}{M_{p1} e^{j\theta_{p1}} M_{pj} e^{j\theta_{pj}} \ldots M_{pn} e^{j\theta_{pn}}}$$

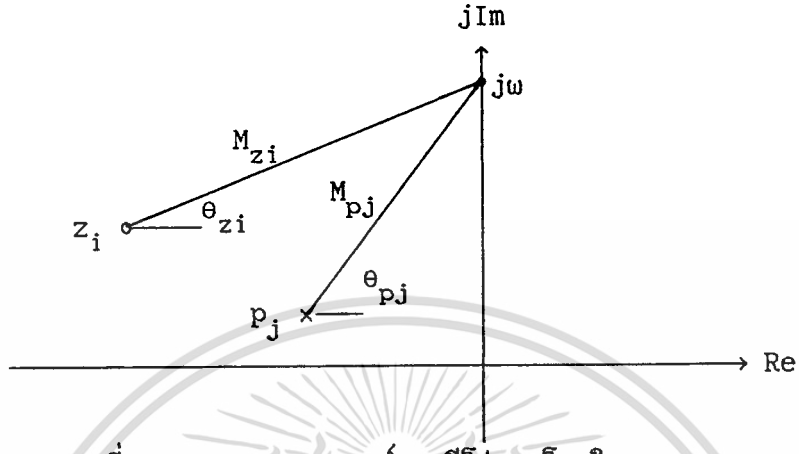
$$= K \frac{M_{z1} M_{zi} \ldots M_{zm}}{M_{p1} M_{pj} \ldots M_{pn}} e^{j(\theta_{z1}+\theta_{zi}+\ldots+\theta_{zm}-\theta_{p1}-\theta_{pj}-\ldots-\theta_{pn})} \qquad (2.5)$$

$$= M(\omega) e^{j\theta(\omega)}$$

เมื่อ $M(\omega)$ เป็นขนาดของ $T(j\omega)$

$\theta(\omega)$ เป็นเฟสของ $T(j\omega)$

ใน S-Plane ค่าของแฟคเตอร์ $(j\omega - z_i)$ สามารถแสดงด้วยเวคเตอร์จาก $z_i$ ไปยัง $j\omega$ และแฟคเตอร์ $(j\omega - p_j)$ สามารถแสดงด้วยเวคเตอร์จาก $p_j$ ไปยัง $j\omega$ ดังนั้นสามารถเขียนเวคเตอร์ซึ่งแสดงขนาด และเฟสได้ดังในรูปที่ 2.1




รูปที่ 2.1 แสดงเวคเตอร์ของซีโร่และโพลใน S-Plane

จากสมการ (2.5) และรูปที่ 2.1 จะเห็นได้ว่าผลตอบสนองทางขนาด $M(\omega)$ คือค่าของ K คูณกับผลคูณของขนาดของเวคเตอร์ที่ลากจากตำแหน่งซีโร่ไปยังความถี่ $\omega$ บนแกนจินตภาพ $M_{zi}$ หารด้วยผลคูณของเวคเตอร์ที่ลากจากตำแหน่งโพลไปยังความถี่ $\omega$ บนแกนจินตภาพ $M_{pj}$ สำหรับผลตอบสนองเฟส $\theta(\omega)$ คือผลรวมของมุมเวคเตอร์ของซีโร่ $\theta_{zi}$ ลบด้วยผลรวมของมุมเวคเตอร์ของโพล $\theta_{pj}$

ผลตอบสนองของกรุ๊ปดีเลย์ (Group Delay) ได้จากการพิจารณาสัญญาณในโดเมนของเวลา โดยแสดงให้อยู่ในรูปของโดเมนของความถี่ เช่นเดียวกันกับผลตอบสนองทางขนาดและเฟส โดยสมมติสัญญาณอินพุท $V_1$ ป้อนไปยังเน็ทเวิร์กที่มีค่ากรุ๊ฟดีเลย์เท่ากับ D วินาที ดังนั้นสัญญาณทางเอาท์พุท $V_2$ จะได้

$$V_2(t) = V_1(t - D) \qquad ...(2.6)$$

เนื่องจากสัญญาณใด ๆ เกิดจากองค์ประกอบของสัญญาณรูปซายน์เสมอ ฉะนั้นสัญญาณอินพุทจะได้

$$V_1 = A \sin(\omega t + \phi) \qquad ...(2.7)$$

จากสมการ (2.6) จะได้สัญญาณเอาท์พุทคือ

$$V_2 = A \sin[\omega(t - D) + \phi]$$

หรือ

$$V_2 = A \sin[\omega t - \omega D + \phi] \qquad ...(2.8)$$

จากสมการ (2.7) และ (2.8) จะเห็นได้ว่าสัญญาณอินพุท และเอาท์พุทมีเฟสต่างกันคือ

$$\theta = -\omega D \qquad ...(2.9)$$

ฟังก์ชั่นกรุ๊ฟดีเลย์ (Group delay) จะได้จากการหาอนุพันธ์ของสมการ (2.9) เทียบกับ $\omega$

ซึ่งจะได้

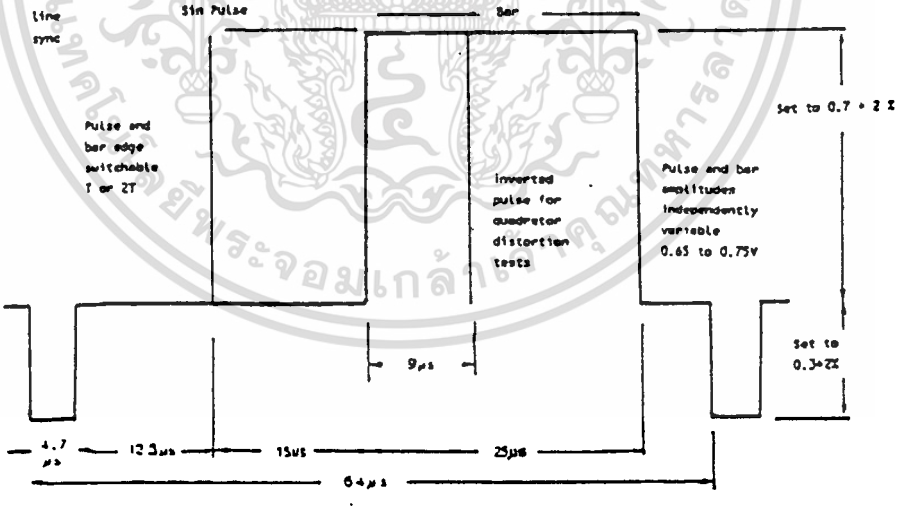
$$D = -\frac{d}{d\omega}\theta \qquad \ldots(2.10)$$

ดังนั้นผลตอบสนองของกรุ๊ฟดีเลย์ก็คือค่าลบการหาอนุพันธ์ผลตอบสนองทางเฟสนั่นเอง

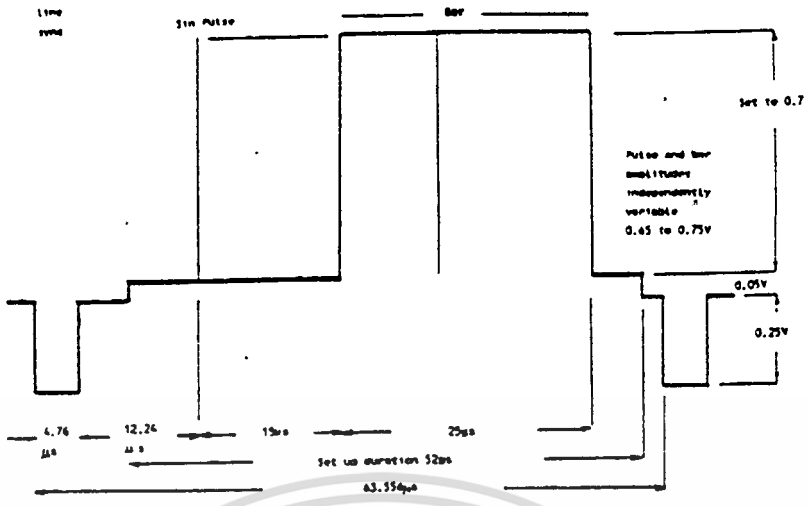
## 2.2 ทฤษฎีพัลซ์ซายน์กำลังสอง

พัลซ์ซายน์กำลังสองเป็นที่นิยมในการใช้ตรวจสอบข้อบกพร่อง ซึ่งอาจจะเกิดขึ้นในระบบโทรทัศน์ [8] พัลซ์ดังกล่าวนี้อาจเป็นแบบ T และ 2T สำหรับตรวจสอบโทรทัศน์ขาว-ดำ และแบบม็อดดูเลท 20 T ซายน์กำลังสองใช้ในการตรวจสอบโทรทัศน์สี วิธีการตรวจสอบระบบใด ๆ ทำได้โดยการป้อนพัลซ์ซายน์กำลังสองให้กับระบบนั้น แล้วสังเกตุการตอบสนองต่าง ๆ ของระบบดังกล่าวคือ การตอบสนองของขนาด (Amplitude Response) การตอบสนองของเฟส (Phase Response)

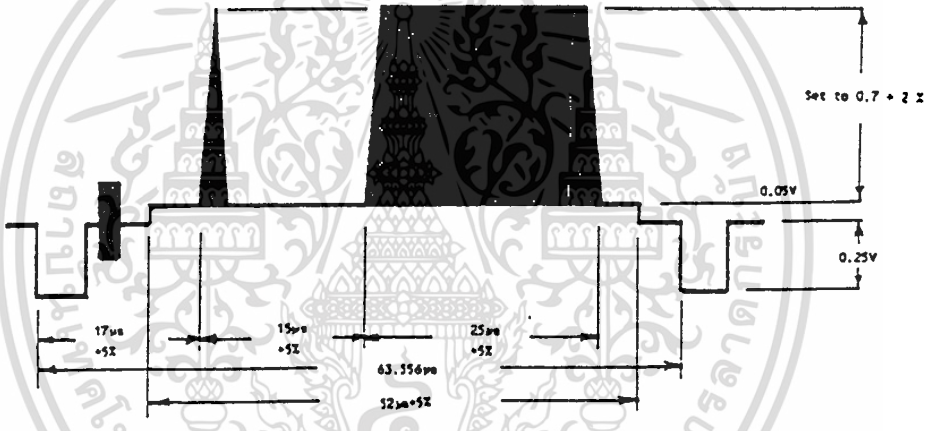
รูปลักษณะคลื่นซายน์กำลังสอง ช่วงกว้างของพัลซ์ที่ใช้ทดสอบมีการกำหนดขนาดของพัลซ์ในช่วงตรงกลาง เช่น ขนาดความกว้างของพัลซ์ช่วงตรงกลาง HAD (Half Amplitude Duration) เมื่อกำหนดให้ HAD เป็นสัญญาณซายน์กำลังสองของ T ช่วงความถี่ของพัลซ์นี้คือ f = 1/T



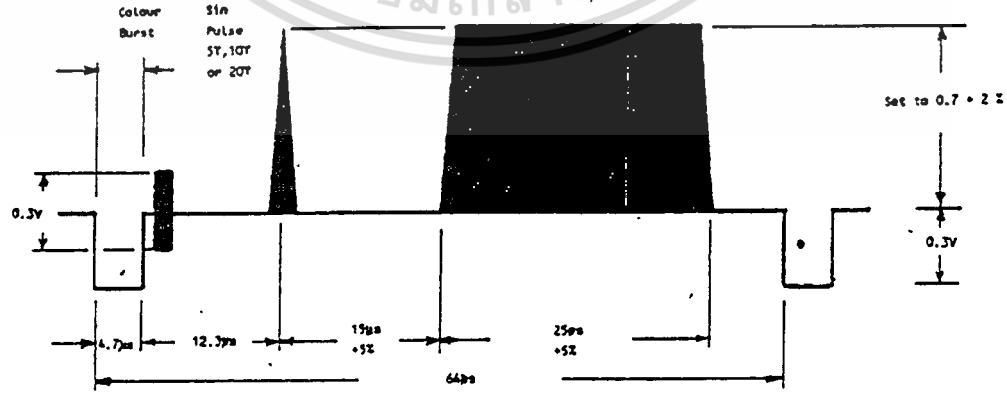

รูปที่ 2.2 รูปลักษณะของสัญญาณพัลซ์ซายน์กำลังสองและบาร์ในระบบ 625 เส้น




รูปที่ 2.3 รูปลักษณะสัญญาณพัลซ์ซายน์กำลังสองและบาร์ในระบบ 525 เส้น




รูปที่ 2.4 รูปลักษณะสัญญาณม็อดดูเลทพัลซ์ซายน์กำลังสองและบาร์ในระบบ 625 เส้น




รูปที่ 2.5 รูปลักษณะสัญญาณม็อดดูเลทพัลซ์ซายน์กำลังสองและบาร์ในระบบ 525 เส้น

ถ้าสัญญาณพัลซ์ซายน์กำลังสองมี HAD เท่ากับ T เราเรียกพัลซ์ดังกล่าวว่า T พัลซ์ ถ้าพัลซ์นั้นมีความกว้างเป็น 2 เท่าเราเรียกว่า 2 T พัลซ์ ถ้ากว้างเป็นครึ่งหนึ่งก็เรียกว่า 1/2 T พัลซ์

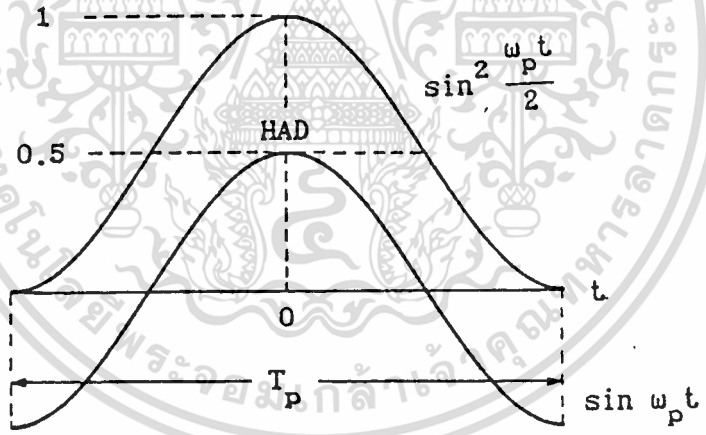
โดยที่สำหรับระบบ NTSC

2 T Pulse มี HAD = 0.25 $\mu$S

T Pulse มี HAD = 0.125 $\mu$S

1/2 T Pulse มี HAD = 0.063 $\mu$S

การใช้พัลซ์ซายน์กำลังสองจำเป็นต้องเข้าใจถึงคุณลักษณะพิเศษของความถี่ใกล้จุดคัท-ออฟ และพัลซ์รูปสี่เหลี่ยม (Bar) ซึ่งเป็นตัวกำหนดย่านความถี่ช่วงต่ำและตรงกลาง ในทางปฏิบัติจะใช้ 2T พัลซ์ , T พัลซ์ และ 1/2 T พัลซ์ เมื่อต้องการตรวจสอบอุปกรณ์โทรทัศน์ขนาดความถี่ 4 MHz เช่นเครื่องบันทึกภาพ (VTR) , อุปกรณ์โทรทัศน์ขนาด 8 MHz และอุปกรณ์โทรทัศน์ที่มีความถี่สูงกว่าตามลำดับ ส่วนสัญญาณสี่เหลี่ยม (Bar) ได้ออกแบบเพื่อให้ตรวจดูได้โดยใช้มอนิเตอร์ เมื่อความถี่ช่วงตรงกลางมีการขยายไม่ดีจะเกิดภาพม้วหรือเป็นริ้วยาว ๆ ดังนั้นสัญญาณสี่เหลี่ยม (Bar) จึงมีประโยชน์มากในการกำหนดคุณลักษณะพิเศษของย่านความถี่ช่วงตรงกลาง




รูปที่ 2.6 แสดงลักษณะของพัลซ์ซายน์กำลังสอง

สมการของพัลซ์ซายน์กำลังสอง ดังแสดงในรูปที่ 2.6 เขียนได้คือ

$$f(t) = \begin{cases} \sin^2 \frac{\omega_p t}{2} \\ \\ \frac{1}{2}[1-\cos \omega_p t] \end{cases} \qquad \ldots(2.11)$$

โดยที่ $\omega_p = 2\pi f_p = 2\pi \frac{1}{T_p}$

เมื่อ $T_p$ เป็นช่วงเวลาความกว้างของพัลซ์

ส่วนสำคัญสำหรับสัญญาณทดสอบพัลซ์ชายน์กำลังสองคือ ช่วงเวลาที่ครึ่งหนึ่งของแอมปริจูด (I) จะมีค่าเท่ากับ 1/2 ของช่วงเวลาของพัลซ์ ($T_p$) ดังนั้น

$$\omega_p = \frac{2\pi}{T_p} = \frac{2\pi}{2T} = \frac{\pi}{T} \qquad ...(2.12)$$

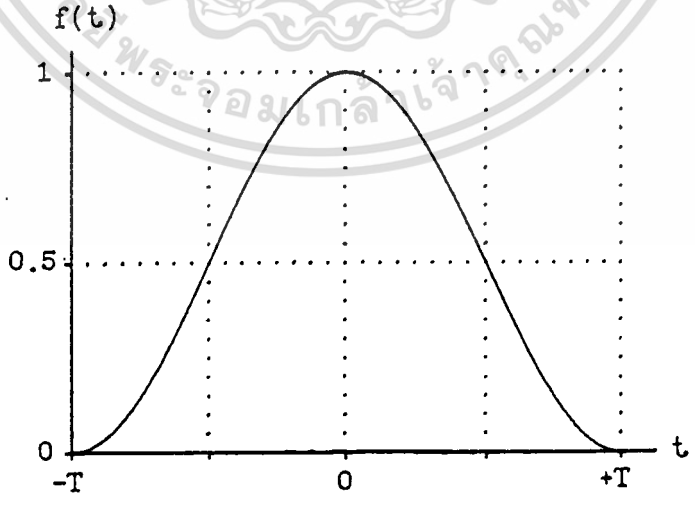
โดยที่ $f_p = \frac{1}{2T}$

ตัวอย่างเช่นในระบบ NTSC สำหรับ T พัลซ์จะมีค่า 0.125 $\mu$S โดยมีสเปคตรัมกว้าง 8 MHz และ 2T พัลซ์จะมีค่า 0.25 $\mu$S ซึ่งจะมีสเปคตรัมกว้าง 4 MHz ส่วนในระบบ PAL 2T พัลซ์จะมีค่า 0.2 $\mu$S ซึ่งคุมย่านความถี่สเปคตรัม 5 MHz

แทนค่า $\omega_p$ จากสมการ (2.12) ลงในสมการ (2.11) จะได้

$$f(t) = \begin{cases} \sin^2 \frac{\pi t}{2T} \\ \frac{1}{2}\left(1-\cos \frac{\pi t}{T}\right) \end{cases} \quad |t| \leq T \qquad ...(2.13)$$

จากสมการ (2.13) สามารถเขียนเป็นพัลซ์ชายน์กำลังสองได้ดังรูปที่ 2.7




รูปที่ 2.7 แสดงชายน์กำลังสองแบบ T พัลซ์

ฟังก์ชั่นของพัลซ์ซายน์กำลังสอง แบบ T, 2T,ม็อดดูเลท 12.5T และม็อดดูเลท 20T หาได้ดังแสดงในตารางที่ 2.1

ตารางที่ 2.1 ฟังก์ชั่นของพัลซ์ซายน์กำลังสองแบบ T,2T,ม็อดดูเลท 12.5T และม็อดดูเลท 20T

| พัลซ์ซายน์กำลังสอง | คอมโพซิทพัลซ์ซายน์กำลังสอง |
|---|---|
| แบบ T<br>$f_1(t) = \begin{cases} \sin^2 \frac{\pi t}{2T} & -T<t<T \\ 0 & \lvert t\rvert>T \end{cases}$ | แบบ 12.5T<br>$f_{12}(t) = f_1(t) + f_2(t)$<br>$f_1(t) = \begin{cases} \frac{1}{2} \sin^2 \frac{\pi t}{25T} & -12.5T<t<12.5T \\ 0 & \lvert t\rvert>12.5T \end{cases}$<br>$f_2(t) = f_1(t) \cos \omega_c t$ |
| แบบ 2T<br>$f_1(t) = \begin{cases} \sin^2 \frac{\pi t}{4T} & -2T<t<2T \\ 0 & \lvert t\rvert>2T \end{cases}$ | แบบ 20T<br>$f_{12}(t) = f_1(t) + f_2(t)$<br>$f_1(t) = \begin{cases} \frac{1}{2} \sin^2 \frac{\pi t}{40T} & -20T<t<20T \\ 0 & \lvert t\rvert>20T \end{cases}$<br>$f_2(t) = f_1(t) \cos \omega_c t$ |

สัญญาณพัลซ์ซายน์กำลังสองมีสมการคือ

$$A_T(t) = \begin{cases} A \sin^2 \frac{\pi t}{2T} & \lvert t\rvert \le T \\ 0 & \lvert t\rvert > T \end{cases} \qquad \ldots(2.14)$$

โดยที่ A เป็นแอมปริจูดของพัลซ์

T เป็นช่วงเวลาฮาฟแอมปริจูดดูเลชั่นของแบบ T พัลซ์

ฟูเรียร์ทรานส์ฟอร์ม

$$A_T(\omega) = \int_{-\infty}^{\infty} A_T(t)\, e^{-j\omega t}\, dt$$

พัลซ์ซายน์กำลังสองเป็นจำนวนคู่ ดังนั้น

$$A_T(\omega) = 2 \int_{0}^{\infty} A_T(t) \cos \omega t\, dt \qquad \ldots(2.15)$$

แทนค่าสมการ (2.14) ลงในสมการ (2.15) จะได้

$$A_T(\omega) = 2\int_0^T A \sin^2 \frac{\pi t}{2T} \cos \omega t \, dt$$

ซึ่งจะได้ฟรีเควันซีสเปคตรัมของแบบ T พัลซ์ ดังนี้

$$A_T(f) = AT \frac{\sin 2\pi fT}{2\pi fT} \left( \frac{1}{1-(2fT)^2} \right) \qquad ...(2.16)$$

ในทำนองเดียวกันฟรีเควันซีสเปคตรัมของพัลซ์แบบ 2T คือ

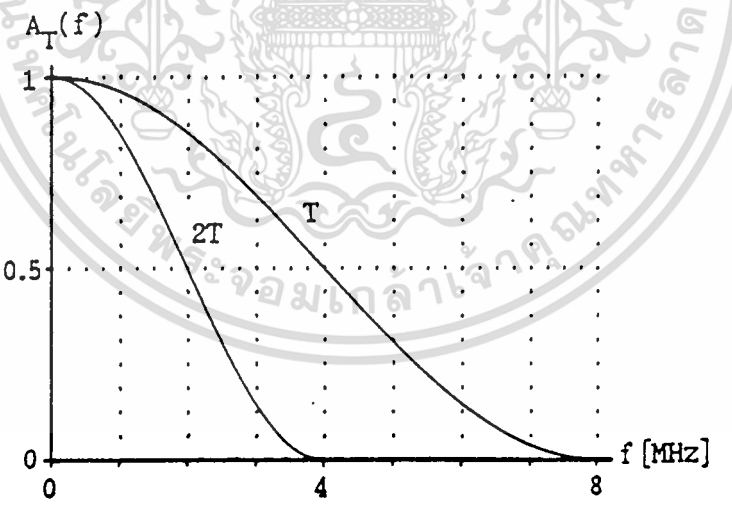
$$A_{2T}(f) = A\ 2T \frac{\sin 4\pi fT}{4\pi fT} \left( \frac{1}{1-(4fT)^2} \right) \qquad ...(2.17)$$

ฟรีเควันซีสเปคตรัมตามอุดมคติของแบบ T พัลซ์ และ 2T พัลซ์ กำหนดได้ดังนี้

$$A_T(f) = \frac{1}{1-(2Tf)^2} \cdot \frac{\sin 2\pi Tf}{2\pi Tf} \qquad ...(2.18)$$

$$A_{2T}(f) = \frac{1}{1-(4Tf)^2} \cdot \frac{\sin 4\pi Tf}{4\pi Tf} \qquad ...(2.19)$$

เอ็นวิโลปของฟรีเควันซีสเปคตรัมของแบบ T พัลซ์ และ 2T พัลซ์ แสดงดังรูปที่ 2.8




รูปที่ 2.8 ฟรีเควันซีสเปคตรัมตามอุดมคติของ T พัลซ์ และ 2T พัลซ์

## 2.3 เกรเดียนท์-เวคเตอร์

เกรเดียนท์-เวคเตอร์ (Gradient-Vector) เป็นทฤษฎีทางคณิตศาสตร์ เหมาะสำหรับใช้ในการคำนวณหาทิศทางใน Design Space เพื่อทำให้สเกลล่าฟังก์ชั่น (Scalar Function) หรือ Cost Function ลดลง โดยมีหลักทฤษฎีว่า

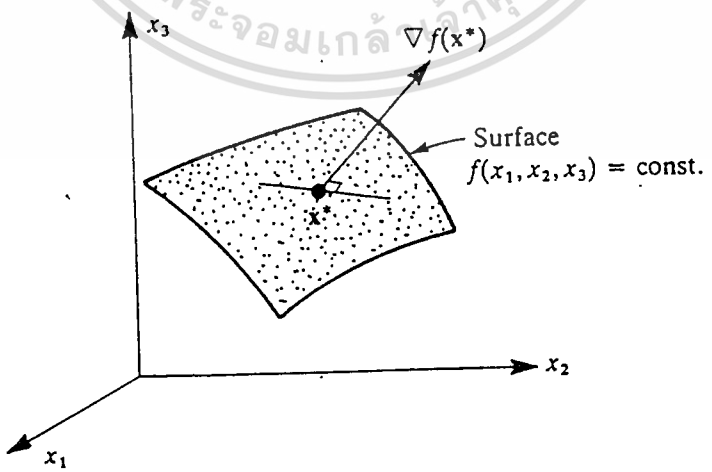
เกรเดียนท์-เวคเตอร์ (Gradient-Vector)ของสเกลล่าฟังก์ชั่น(Scalar Function) ที่ใช้ในการออกแบบหาจุด Optimum [20] โดยพิจารณาฟังก์ชั่น f(x) ที่มีตัวแปร n ตัว คือ $x_1, x_2, \ldots, x_n$ พาร์เชี่ยน ดีริเวอร์ทีฟ (Partial Derivative) ของฟังก์ชั่น f(x) ที่กระทำกับ $x_1$ ที่จุด $x^*$ คือ $\partial f(x^*)/\partial x_1$ ,ฟังก์ชั่นที่กระทำกับ $x_2$ ที่จุด $x^*$ คือ $\partial f(x^*)/\partial x_2$ และจนถึง $x_n$ ให้ $\bar{C}_i$ แทนด้วย พาร์เชียร์ ดีริเวอร์ทีฟของ f(x) กระทำกับ $x_i$ ที่จุด $x^*$ ซึ่งสามารถเขียนได้ดัง

$$\nabla f = \left[ \frac{\partial f}{\partial x_1} \; \frac{\partial f}{\partial x_2} \; \frac{\partial f}{\partial x_3} \; \cdots \; \frac{\partial f}{\partial x_n} \right]^T = \bar{C} \qquad \ldots(2.45)$$

สมการ (2.45) แยกเขียนได้เป็น $\partial f(x^*)/\partial x_1$ , $\partial f(x^*)/\partial x_2$ ,..., $\partial f(x^*)/\partial x_n$ ซึ่งเรียกว่า เกรเดียนท์-เวคเตอร์ แทนด้วยสัญลักษณ์ $\bar{C}, \nabla f, \partial f/\partial x$ และ Grad ของฟังก์ชั่น เพราะฉะนั้น เกรเดียนท์ของฟังก์ชั่น f(x) ที่จุด $x^*$ สามารถเขียนทางคอลลัมเวคเตอร์ (Column Vector) ได้

$$\bar{C} = \bar{C}(x_i) = \left[ \frac{\partial f(x)}{\partial x_i} \right] \qquad \ldots(2.46)$$

T เป็น ทรานส์โพส (Transpose) ของ โร-เวคเตอร์ (Row Vector) โดยปกติ เกรเดียนท์-เวคเตอร์จะเป็นแทนเจนท์เพลน (Tangent Plant) ที่จุด $x^*$ ดังแสดงในรูป 2.15

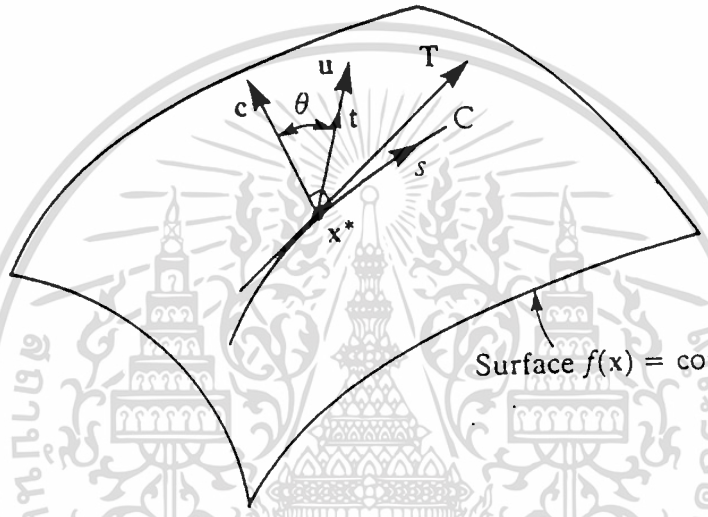



รูปที่ 2.15 แสดงเกรเดียนท์-เวคเตอร์ที่ตั้งฉากกับแทนเจนท์เพลน ที่จุด $x^*$

ในกรรมวิธี Optimization นั้นอาศัยคุณสมบัติแต่ละข้อของเกรเดียนท์-เวคเตอร์ มาออกแบบหาจุดที่ดีที่สุด ซึ่งคุณสมบัติแต่ละข้อนั้นสามารถจะพิสูจน์ได้ต่อไป

**คุณสมบัติข้อ 1.**

เกรเดียนท์-เวคเตอร์ $\bar{C}$ ของฟังก์ชั่น $f(x_1,x_2,\ldots x_n)$ ที่จุด $x^* = (x_1^*,x_2^*,\ldots x_n^*)$ โดยปกติจะต้องตั้งฉากกับแทนเจนท์เพลนของผิวสัมผัสซึ่งฟังก์ชั่น $f(x_1,x_2,\ldots,x_n)$ มีค่าเท่ากับค่าคงที่




รูปที่ 2.16 แสดงเกรเดียนท์-เวคเตอร์สำหรับพื้นผิวสัมผัส $f(x)$ = ค่าคงที่ๆจุด $x^*$

คุณสมบัติที่สำคัญของเกรเดียนท์-เวคเตอร์ แสดงในรูป(2.16)จากรูปแสดง พื้นผิวสัมผัสของ $f(x)$ = ค่าคงที่, $x^*$ เป็นจุดๆหนึ่งบนพื้นผิวสัมผัส, c เป็นพื้นผิวสัมผัสที่ผ่านจุด $x^*$,$\bar{T}$ เป็นเวคเตอร์แทนเจนท์เพลน ของส่วนโค้ง c ที่จุด $x^*$, $\bar{U}$ เป็น ยูนิทเวคเตอร์ ใดๆ และ $\bar{C}$ เป็นเกรเดียนท์-เวคเตอร์ที่จุด $x^*$ โดยปกติ $\bar{C}\cdot\bar{T} = 0$

**พิสูจน์คุณสมบัติข้อ 1.**

จากรูป 2.16 ให้ทุกๆส่วนโค้ง c บนพื้นผิวสัมผัสของ $f(x_1,x_2,\ldots x_n)$ เท่ากับค่าคงที่, ให้ส่วนโค้ง c ผ่านจุด $x^* = (x_1^*,x_2^*,\ldots x_n^*)$ และให้ S เป็นพื้นผิวตาม c จากนั้นให้แทนเจนท์เวคเตอร์ $\bar{T}$ หนึ่งหน่วยตาม c ที่จุด $x^*$ มีสมการดัง (2.47)

$$\bar{T} = \left[ \frac{\partial x_1}{\partial s} \quad \frac{\partial x_2}{\partial s} \quad \frac{\partial x_3}{\partial s} \quad \cdots \quad \frac{\partial x_n}{\partial s} \right]^T \qquad \ldots(2.47)$$

เนื่องจากฟังก์ชั่น f(x) = ค่าคงที่, ดีริเวอร์ทีฟ ของฟังก์ชั่นตามส่วนโค้ง c เป็นศูนย์

$$\frac{df}{ds} = 0 \qquad \ldots(2.48)$$

ใช้กฎลูกโซ่ (Chain Rule) ของการ ดิฟเฟอร์เรนเชียล (Differentiation)

$$\frac{df}{ds} = \left[ \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial s} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial s} + \cdots + \frac{\partial f}{\partial x_n}\frac{\partial x_n}{\partial s} \right]^T \qquad \ldots(2.49)$$

จากสมการ (2.49) จะได้ $\partial f/\partial x_1$ , $\partial f/\partial x_2$ และสมการ (2.47)แต่ละส่วนของเกรเดียนท์และยูนิทแทนเจนท์เวคเตอร์ จะได้

$$\bar{C} \cdot \bar{T} = 0 \qquad \ldots(2.50)$$

แต่ดอท (Dot) ของเกรเดียนท์-เวคเตอร์ $\bar{C}$ กับแทนเจนท์เวคเตอร์ $\bar{T}$ เป็น 0 และ $\bar{T}$ เป็นแทนเจนท์เวคเตอร์ที่จุด $x^*$ ดังนั้น $\bar{C}$ ตั้งฉากกับ $\bar{T}$ สำหรับพื้นผิวสัมผัสของฟังก์ชั่น f(x) = ค่าคงที่ๆจุด $x^*$

**คุณสมบัติข้อ 2.**

ค่าของเกรเดียนท์ แทนด้วยทิศทาง อัตราสูงสุด (Maximium Rate) สำหรับการเพิ่มขึ้นของฟังก์ชั่น f(x) ที่จุด $x^*$

**พิสูจน์คุณสมบัติข้อ 2.**

ให้ $\bar{U}$ เป็นยูนิทเวคเตอร์ ในทุกๆทิศทาง แต่ไม่ใช่แทนเจนท์ ที่พื้นผิวสัมผัส แสดงในรูป 2.16, ให้ t เป็นพารามิเตอร์ (Parameter) ตามทิศทาง $\bar{U}$ ดิริเวอร์ทีฟของฟังก์ชั่น f(x) ในทิศทาง $\bar{U}$ ที่จุด $x^*$ เป็นดัง

$$\frac{df}{dt} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon\mu) - f(x)}{\epsilon} \qquad \ldots(2.51)$$

เมื่อ $\epsilon$ เป็นค่าเล็กๆ ใช้เทร์เลอร์ ซีรี่ (Taylor Series) กระจายจะได้

$$f(x+\epsilon\mu) = f(x) + \epsilon\left[\mu_1\frac{\partial f}{\partial x_1} + \mu_2\frac{\partial f}{\partial x_2} + \cdots + \mu_n\frac{\partial f}{\partial x_n}\right] + O(\epsilon^2) \quad ...(2.52)$$

เมื่อ $\bar{U}_i$ เป็นแต่ละส่วนของยูนิทเวคเตอร์ $\bar{U}$ และ $O(\epsilon^2)$ เป็นเทอม (Term) ของอันดับสองจากสมการสามารถเขียนได้ใหม่เป็น

$$f(x+\epsilon\mu) - f(x) = \epsilon \sum_{i=1}^{n} \mu_i\frac{\partial f}{\partial x_i} + O(e^2) \quad ...(2.53)$$

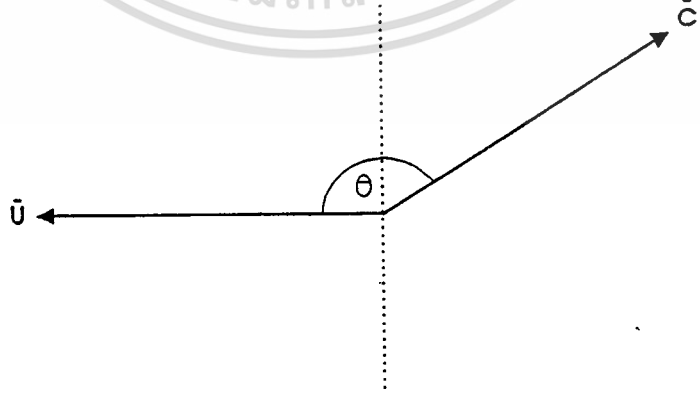
แทนสมการ (2.53) ลงในสมการ (2.52) และทำการลิมิต (Limit) $\epsilon$ เข้าใกล้ศูนย์จะได้

$$\frac{df}{dt} = \sum_{i=1}^{n} \mu_i\frac{\partial f}{\partial x_i} = \bar{C}\cdot\mu = \bar{C}^T\cdot\mu \quad ...(2.54)$$

ใช้ทฤษฎีของ ดอท โพรดัท (Dot Product) จะได้

$$\frac{df}{dt} = |\bar{C}| \cdot |\bar{U}| \quad ...(2.55)$$

เมื่อ $\theta$ เป็นมุมระหว่างเวคเตอร์ $\bar{C}$ และเวคเตอร์ $\bar{U}$ ถ้า $\theta = 0^\circ$ เวคเตอร์ $\bar{U}$ จะไปตามทิศทางของเวคเตอร์ $\bar{C}$ จะได้ $\cos\theta=1$ เพราะฉะนั้นจากสมการ (2.55) df/dt จะแทนด้วยอัตราการเพิ่มสูงสุดของฟังก์ชั่น $f(x)$ ในทำนองเดียวกันถ้า $\theta = 180^\circ$ เวคเตอร์ $\bar{U}$ จะอยู่ในทิศทางตรงกันข้ามกับเวคเตอร์ $\bar{C}$ จะได้ $\cos 180^\circ = -1$ เพราะฉะนั้นจากสมการ (2.55) df/dt แทนด้วยอัตราการลดลงสูงสุดของฟังก์ชั่น $f(x)$ ดังรูป 2.17




รูปที่ 2.17 แสดงเวคเตอร์ $\bar{C}$ กระทำกับเวคเตอร์ $\bar{U}$ เป็นมุม $\theta$

จากคุณสมบัติข้อที่ 2 ของเกรเดียนท์-เวคเตอร์ ถ้าเคลื่อนทิศทางของฟังก์ชั่น f(x) ไปตามทิศทางเกรเดียนท์-เวคเตอร์ ฟังก์ชั่น f(x) จะเพิ่มขึ้นอย่างรวดเร็วเมื่อเทียบกับการเคลื่อนที่ไปตามทิศทางอื่นๆ เช่นเดียวกันถ้าเคลื่อนทิศทางของฟังก์ชั่น f(x) ไปตามทิศทางตรงข้ามกับเกรเดียนท์-เวคเตอร์ (เน็คกาทีฟ-เกรเดียนท์) ฟังก์ชั่น f(x) จะลดลงอย่างรวดเร็ว และถ้าเคลื่อนทิศทางของฟังก์ชั่น f(x) ไปตามทิศทางของแทนเจนท์เพลน ผลก็คือจะไม่มีการเปลี่ยนแปลงของฟังก์ชั่น f(x)

**คุณสมบัติข้อ 3.**

อัตราการเปลี่ยนแปลงของฟังก์ชั่น f(x) ที่ทุกๆจุด $x^*$ เป็นขนาดของ เกรเดียนท์-เวคเตอร์ เนื่องจาก $\bar{U}$ เป็น ยูนิท-เวคเตอร์ ค่าสูงสุดของ df/dt จากสมการ (2.55) จะได้

$$\text{Max } \frac{df}{dt} = |\bar{C}| \qquad \ldots(2.56)$$

สำหรับ $\theta = 0^\circ$ $\bar{U}$ จะอยู่ในทิศทางของเกรเดียนท์-เวคเตอร์ เพราะฉะนั้นขนาดของเกรเดียนท์-เวคเตอร์ แทนด้วยอัตราการเพิ่มสูงสุดของการเปลี่ยนแปลงสำหรับฟังก์ชั่น f(x)

จากคุณสมบัติข้อที่ 3 ของเกรเดียนท์-เวคเตอร์นี้แสดงว่า เกรเดียนท์-เวคเตอร์ที่ทุกๆจุดของ $x^*$ แทนด้วยทิศทางหนึ่งๆของการเพิ่มขึ้นสูงสุดของฟังก์ชั่น f(x) และอัตราการเพิ่มเป็นขนาดของเกรเดียนท์-เวคเตอร์

# บทที่ 3
# การประมาณฟังก์ชั่นด้วยวิธี ทรานซิทชั่นนัลอุลตร้าสเฟียริคัล-ทอมป์สันโพลิโนเมียล

การประมาณฟังก์ชั่นในขอบช่ายเวลา เพื่อให้ได้เน็ทเวิกร์ฟังก์ชั่นที่ตอบสนองต่อสัญญาณอิมพัลซ์ ซึ่งต้องอาศัยรูปแบบวิชาคณิตศาสตร์ (Calculus, Matrix Algebra, Laplace Transform, Differential Equation) และใช้เทคนิคในการประมาณฟังก์ชั่น โดยใช้วิธี ลีซ-ซแคว (Least-Squares Method) [12] เพื่อคำนวณหาค่าสัมประสิทธิ์ในเทอมของตัวแปรต่างๆสำหรับเน็ทเวิกร์ฟังก์ชั่นที่ให้ผลตอบสนองในขอบช่ายเวลา ซึ่งวิธี ลีซ-ซแคว นี้เราจะต้องทราบค่าโพลของเน็ทเวิกร์ฟังก์ชั่นมาก่อน จึงจะสามารถออกแบบการประมาณฟังก์ชั่นได้ โพลที่ทราบค่านี้หาได้จากทรานซิทชั่นนัลอุลตร้าสเฟียริคัล-ทอมป์สันโพลิโนเมียล ซึ่งการทรานส์ซิทชั่นนัลนี้เป็นการหาตำแหน่งโพล (Relocate Pole) ให้เหมาะสม เพื่อให้การประมาณฟังก์ชั่นมีผลตอบสนองในขอบช่ายเวลาที่ใกล้เคียงกับฟังก์ชั่นอุดมคติ

## 3.1 วงจรกรองความถี่ต่ำแบบอุลตร้าสเฟียริคัลโพลิโนเมียล

วงจรกรองความถี่ต่ำแบบอุลตร้าสเฟียริคัลโพลิโนเมียล มีทรานส์เฟอร์ฟังก์ชั่นซึ่งเขียนอยู่ในรูปของขนาดกำลังสอง (Magnitude Square) คือ

$$H(S).H(-S)|_{s=j\omega} = |H_n(j\omega)|^2 = \frac{1}{1 + U_n^{\alpha}(\omega^2)} \qquad ...(3.1)$$

โดยที่ $U_n^{\alpha}(\omega^2)$ เป็นขนาดกำลังสอง (Magnitude Square) ของอุลตร้าสเฟียริคัลโพลิโนเมียล และ n เป็นอันดับของทรานส์เฟอร์ฟังก์ชั่น

สูตรทั่วไปของอุลตร้าสเฟียริคัลโพลิโนเมียล [13] $U_n^{\alpha}(\omega)$ ที่ใช้คือ

$$U_n^{\alpha}(\omega) = \frac{n!}{(1 + \alpha)_n} P_n^{(\alpha,\alpha)}(\omega) \qquad ...(3.2)$$

โดยที่ $(1+\alpha)_n = (1+\alpha)(2+\alpha).....(n+\alpha)$ ...(3.3)

$n = 1, 2, 3, ...; \quad \alpha > -1$

$\alpha$ เป็นค่าพารามิเตอร์ของอุลตร้าสเฟียริคัลโพลิโนเมียล และโพลิโนเมียล $P_n^{(\alpha,\alpha)}(\omega)$ ได้มาจาก Jacobi โพลิโนเมียล โดยที่ Jacobi โพลิโนเมียล $P_n^{(\alpha,\beta)}(\omega)$ กำหนดได้ดังนี้ [13]

$$P_n^{(\alpha,\beta)}(\omega) = \frac{(1+\alpha)_n}{n!}\ {}_2F_1\left(\begin{matrix}-n,\alpha+\beta+n+1 \\ 1+\alpha\end{matrix};\ \frac{1-\omega}{2}\right) \qquad ...(3.4)$$

โดยที่ ${}_2F_1$ เป็น Hypergeometric ฟังก์ชั่น [13] ซึ่งมีสูตรทั่วไปดังนี้

$${}_2F_1\left(\begin{matrix}a,b \\ c\end{matrix};\ x\right) = \sum_{k=0}^{n} \frac{(a)_k\,(b)_k\,x^k}{k!\,(c)_k} \qquad ...(3.5)$$

เมื่อ $\beta = \alpha = 0$ ค่าโพลิโนเมียลในสมการ (3.4) กลายเป็น Legendre โพลิโนเมียล จากสมการ (3.4) จะเห็นได้ว่า ค่า $P_n^{(\alpha,\beta)}(\omega)$ เป็นโพลิโนเมียลของอันดับที่ n และเมื่อ $\omega = 1$ จะได้

$$P_n^{(\alpha,\beta)}(\omega) = P_n^{(\alpha,\beta)}(1) = \frac{(1+\alpha)_n}{n!} \qquad ...(3.6)$$

ในกรณีของ Jacobi โพลิโนเมียล ค่าของ ${}_2F_1$ สามารถประยุกต์ได้ดังสมการต่อไปนี้

$$P_n^{(\alpha,\beta)}(\omega) = \frac{(1+\alpha)_n}{n!}\left(\frac{\omega+1}{2}\right)^n {}_2F_1\left(\begin{matrix}-n,-\beta+n \\ 1+\alpha\end{matrix};\ \frac{\omega-1}{\omega+1}\right) \qquad ...(3.7)$$

$$P_n^{(\alpha,\beta)}(\omega) = \frac{(-1)^n\,(1+\beta)_n}{n!}\ {}_2F_1\left(\begin{matrix}-n,\beta+\alpha+n+1 \\ 1+\beta\end{matrix};\ \frac{1+\omega}{2}\right) \qquad ...(3.8)$$

จากสมการ (3.4) , (3.7) และ (3.8) ถ้าให้ผลลัพธ์อยู่ในรูปของ Finife Series สำหรับค่า $P_n^{(\alpha,\beta)}(\omega)$ จะเขียนได้ใหม่ดังสมการ (3.9),(3.10) และ (3.11)

$$P_n^{(\alpha,\beta)}(\omega) = \sum_{k=0}^{n} \frac{(1+\alpha)_n\,(1+\alpha+\beta)_{n+k}}{k!(n-k)!(1+\alpha)_k(1+\alpha+\beta)_n}\left(\frac{\omega-1}{2}\right)^k \qquad ...(3.9)$$

$$P_n^{(\alpha,\beta)}(\omega) = \sum_{k=0}^{n} \frac{(1+\alpha)_n\,(1+\beta)_n}{k!(n-k)!(1+\alpha)_k(1+\beta)_{n-k}}\left(\frac{\omega-1}{2}\right)^k\left(\frac{\omega+1}{2}\right)^{n-k} \qquad ...(3.10)$$

$$P_n^{(\alpha,\beta)}(\omega) = \sum_{k=0}^{n} \frac{(-1)^{n-k}(1+\beta)_n(1+\alpha+\beta)_{n+k}}{k!(n-k)!(1+\beta)_k(1+\alpha+\beta)_n} \left(\frac{\omega+1}{2}\right)^k \quad ...(3.11)$$

ในกรณีที่ใช้กับสมการที่มี order มาก จากสมการ (3.9),(3.10) และ (3.11) สามารถเขียนใหม่ได้ดังสมการ (3.12),(3.13) และ (3.14) ตามลำดับ

$$P_n^{(\alpha,\beta)}(\omega) = \frac{(1+\alpha+\beta)_{2n}}{n!(1+\alpha+\beta)_n} \left(\frac{\omega-1}{2}\right)^n {}_2F_1\left(\begin{matrix}-n, -\alpha-n \\ -\alpha-\beta-2n\end{matrix}; \frac{2}{1-\omega}\right) \quad ...(3.12)$$

$$P_n^{(\alpha,\beta)}(\omega) = \frac{(1+\beta)_n}{n!} \left(\frac{\omega-1}{2}\right)^n {}_2F_1\left(\begin{matrix}-n, -\alpha-n \\ 1+\beta\end{matrix}; \frac{\omega+1}{\omega-1}\right) \quad ...(3.13)$$

$$P_n^{(\alpha,\beta)}(\omega) = \frac{(1+\alpha+\beta)_{2n}}{n!(1+\alpha+\beta)_n} \left(\frac{\omega+1}{2}\right)^n {}_2F_1\left(\begin{matrix}-n, -\beta-n \\ -\alpha-\beta-2n\end{matrix}; \frac{2}{\omega+1}\right) \quad ...(3.14)$$

จากสมการข้างต้นเป็นความสัมพันธ์ของฟังก์ชั่น Jacobi โพลิโนเมียล และ ถ้ากำหนดให้ $\beta = \alpha$ แล้วจากสมการ (3.4) จะได้โพลิโนเมียล $P_n^{(\alpha,\alpha)}(\omega)$ ดังนี้

$$P_n^{(\alpha,\alpha)}(\omega) = \frac{(1+\alpha)_n}{n!} {}_2F_1\left(\begin{matrix}-n, 2\alpha+n+1 \\ 1+\alpha\end{matrix}; \frac{1-\omega}{2}\right) \quad ...(3.15)$$

ฉะนั้นเมื่อแทนค่า Hypergeometric ฟังก์ชั่น ${}_2F_1$ ในสมการ (3.5) ลงในสมการ (3.15) จะได้

$$P_n^{(\alpha,\alpha)}(\omega) = \frac{(1+\alpha)_n}{n!} \sum_{k=0}^{n} \frac{(-n)_k\ (2\alpha+n+1)_k}{k!\ (1+\alpha)_k} \left(\frac{1-\omega}{2}\right)^k \quad ...(3.16)$$

ฉะนั้นถ้าแทนค่าสมการ (3.16) ลงในสมการ (3.2) จะได้

$$U_n^{\alpha}(\omega) = \sum_{k=0}^{n} \frac{(-n)_k\ (2\alpha+n+1)_k}{k!\ (1+\alpha)_k} \left(\frac{1-\omega}{2}\right)^k \quad ...(3.17)$$

โดยที่ $$(-n)_k = (-1)^k \frac{n!}{(n-k)!} \quad ...(3.18)$$

โดยการแทนค่าสมการ (3.18) ลงในสมการ (3.17) จะได้

$$U_n^{\alpha}(\omega) = \sum_{k=0}^{n} (-1)^k C_k^n \frac{(2\alpha+n+1)_k}{(1+\alpha)_k} \left(\frac{1-\omega}{2}\right)^k \quad ...(3.19)$$

โดยที่ $C_k^n$ เป็นสัมประสิทธิ์ของ Binomial ซึ่งเท่ากับ $\frac{n!}{(n-k)!k!}$

จากสมการ (3.19) ถ้า n = 1,2,3,4 และ 5 ค่า $U_n^{\alpha}(\omega)$ เขียนเป็นตารางได้ดังแสดงในตารางที่ 3.1

| n | $U_n^{\alpha}(\omega)$ |
|---|---|
| 0 | $1$ |
| 1 | $\omega$ |
| 2 | $\frac{(2\alpha+3)\omega^2 - 1}{2\alpha + 2}$ |
| 3 | $\frac{(2\alpha+5)\omega^3 - 3\omega}{2\alpha + 2}$ |
| 4 | $\frac{(4\alpha^2+24\alpha+35)\omega^4 - (12\alpha+30)\omega^2 + 3}{4\alpha^2 + 12\alpha + 8}$ |
| 5 | $\frac{(8\alpha^3+84\alpha^2+286\alpha+315)\omega^5 - (40\alpha^2+240\alpha+350)\omega^3 + (30\alpha+75)\omega}{8\alpha^3 + 44\alpha^2 + 76\alpha + 40}$ |

ตารางที่ 3.1 แสดงค่าอุลตร้าสเฟียริคัลโพลิโนเมียล

ถ้ากำหนดให้ n = 4 เปลี่ยนค่า $\alpha$ ตั้งแต่ -0.2 ถึง 1 จะได้ค่า $U_4^{\alpha}(\omega)$ ดังแสดงในตารางที่ 3.2

| $\alpha$ | $U_4^{\alpha}(\omega)$ |
|---|---|
| -0.2 | $5.2708\omega^4 - 4.7916\omega^2 + 0.5208$ |
| 0 | $4.375\omega^4 - 3.75\omega^2 + 0.375$ |
| 0.2 | $3.784\omega^4 - 3.0681\omega^2 + 0.2841$ |
| 0.4 | $3.3661\omega^4 - 2.5892\omega^2 + 0.2232$ |
| 0.6 | $3.0552\omega^4 - 2.2355\omega^2 + 0.1802$ |
| 0.8 | $2.8154\omega^4 - 1.9642\omega^2 + 0.1488$ |
| 1 | $2.625\omega^4 - 1.75\omega^2 + 0.125$ |

ตารางที่ 3.2 แสดงค่า $U_4^{\alpha}(\omega)$ โดยการกำหนดค่า $\alpha$

จากค่า $U_4^{\alpha}(\omega)$ ในตารางที่ 3.2 แทนลงในสมการ (3.1) ได้ทรานส์เฟอร์ฟังก์ชั่นของอุลตร้าสเฟียริคัลโพลิโนเมียล ที่อยู่ทางด้านซ้ายมือของ S-Plane ดังแสดงในตารางที่ 3.3

| $\alpha$ | โพลิโนเมียล |
|---|---|
| -0.2 | $S^4 + 0.8671S^3 + 1.2850S^2 + 0.6083S + 0.3478$ |
| 0 | $S^4 + 1.0195S^3 + 1.3768S^2 + 0.7247S + 0.2441$ |
| 0.2 | $S^4 + 1.1471S^3 + 1.4688S^2 + 0.8278S + 0.2747$ |
| 0.4 | $S^4 + 1.2553S^3 + 1.5571S^2 + 0.9197S + 0.3043$ |
| 0.6 | $S^4 + 1.3480S^3 + 1.6403S^2 + 1.0023S + 0.3325$ |
| 0.8 | $S^4 + 1.4284S^3 + 1.7179S^2 + 1.0770S + 0.3590$ |
| 1 | $S^4 + 1.4989S^3 + 1.7900S^2 + 1.1449S + 0.3839$ |

ตารางที่ 3.3 แสดงค่าอุลตร้าสเฟียริคัลโพลิโนเมียล

## 3.2 วงจรกรองความถี่ต่ำแบบทอมป์สันโพลิโนเมียล

วงจรกรองความถี่ต่ำแบบทอมป์สันโพลิโนเมียล ซึ่งหาโพลิโนเมียลได้จากเบสเซลโพลิโนเมียล [15] ได้ดังสมการที่ (3.20)

$$B_n(s) = \sum_{k=0}^{n} b_k s^k \qquad \ldots(3.20)$$

โดยที่ $b_k$ เป็น ส.ป.ส ที่ใดๆจาก 0 ถึง n และ n เป็นจำนวนอันดับของเบสเซลโพลิโนเมียล จากสมการที่ (3.20) ค่า $b_k$ หาได้จากสมการ (3.21)

$$b_k = \frac{(2n-k)!}{2^{(n-k)}k!(n-k)!} \qquad \ldots(3.21)$$

แทนค่า $b_k$ ลงในสมการที่ (3.20) จะได้สมการหาค่าเบสเซลโพลิโนเมียลที่อันดับต่างๆ ดังสมการที่ (3.22)

$$B_n(s) = \sum_{k=0}^{n} \frac{(2n-k)!s^k}{2^{(n-k)}(n-k)!k!} \qquad \ldots(3.22)$$

จากสมการ (3.22) ถ้า n=1,2,3 และ 4 จะได้ค่าทอมป์สันโพลิโนเมียลดังตารางที่ 3.4

| n | $B_n(s)$ |
|---|---|
| 1 | $s + 1$ |
| 2 | $s^2 + 3s + 3$ |
| 3 | $s^3 + 6s^2 + 15s + 15$ |
| 4 | $s^4 + 10s^3 + 45s^2 + 105s + 105$ |

ตารางที่ 3.4 แสดงค่าทอมป์สันโพลิโนเมียลที่อันดับ 1 ถึง 4

## 3.3 วงจรกรองความถี่แบบบัตเตอร์เวิธ

วงจรกรองความถี่แบบบัตเตอร์เวิธมีทรานส์เฟอร์ฟังก์ชั่น ซึ่งเขียนในรูปของขนาดกำลังสอง (Magnitude Square) [14] คือ

$$H(S).H(-S)|_{s=j\omega} = |H_n(j\omega)|^2 = \frac{1}{1 + \omega^{2n}} \quad ...(3.23)$$

โดยที่ n เป็นลำดับ (order) ของทรานส์เฟอร์ฟังก์ชั่น หรือโพลิโนเมียล

ถ้ากำหนดให้ loss = $|B_n(j\omega)|^2$

$$|B_n(j\omega)|^2 = \frac{1}{|H_n(j\omega)|^2} = 1 + \omega^{2n} \quad ...(3.24)$$

จากสมการ (3.24) แทน $S = j\omega$ ; $\omega = -Sj$ จะได้

$$|B_n(j\omega)|^2 = 1 + (-S^2)^n \quad ...(3.25)$$

จากสมการ (3.25) หารากของสมการได้โดย

$$1 + (-1)^n(S^{2n}) = 0$$

$$(-1)^n S^{2n} = -1 = e^{j(2c-1)\pi} \quad ...(3.26)$$

โดยที่ $(-1)^n = e^{jn\pi}$ จะได้

$$S^{2n} = e^{j(2c+n-1)\pi} \quad ...(3.27)$$

ฉะนั้นรากของสมการคือ

$$S_c = e^{j[\pi/2.(2c+n-1)/n]} \quad ...(3.28)$$

โดยที่ $c = 1, 2, ..., n$

ตัวอย่าง ถ้า n = 2 เราสามารถหาร่วมจากสมการ (3.28) ได้ดังนี้

$$H_2(S) = \frac{1}{(S+S_1)(S+S_2)}$$

$$S_1 = e^{j3\pi/4} = -0.707 + j0.707$$

$$S_2 = e^{j5\pi/4} = -0.707 - j0.707$$

$$H_2(S) = \frac{1}{(S+0.707+j0.707)(S+0.707+j0.707)}$$

$$= \frac{1}{S^2+1.414S+1} \qquad \ldots(3.29)$$

จากสมการ (3.29) เป็นทรานส์เฟอร์ฟังก์ชั่นของวงจรกรองความถี่ต่ำแบบบัตเตอร์เวิร์ท เมื่อกำหนดให้ n = 2 และจากสมการ (3.27) ถ้าแทน n = 2 ถึง 4 จะได้สัมประสิทธิ์ของโพลิโนเมียลดังแสดงในตารางที่ 3.5

| n | $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
|---|---|---|---|---|---|---|
| 2 | 1 | 1.4142 | | | | |
| 3 | 1 | 2 | 2 | | | |
| 4 | 1 | 2.6131 | 3.4142 | 2.6131 | | |

ตารางที่ 3.5 แสดงสัมประสิทธิ์ของบัตเตอร์เวิรโพลิโนเมียล $B_n(S) = S^n + \sum_{i=0}^{n-1} a_i S^i$

## 3.4 การออกแบบวงจรกรองความถี่ต่ำทรานสชิทชั่นนัลอุลตร้าสเฟียริคัล-ทอมสันและบัตเตอร์เวิร์ท-ทอมป์สันโพลิโนเมียล

ในการออกแบบวงจรกรองความถี่ต่ำทรานสชิทชั่นนัลอุลตร้าสเฟียริคัล-ทอมสัน (TUT) และบัตเตอร์เวิร์ท-ทอมป์สัน (TBT) โพลิโนเมียล กำหนดให้ทรานส์เฟอร์ฟังก์ชั่น

$$H_n(S) = \frac{1}{S^n + b_{n-i}S^{n-i} + ..... + b_1S+1} \quad ...(3.30)$$

โดยที่ $i = 1,2,...,n-1$

จากสมการที่ (3.30) เขียนใหม่ได้ดังสมการที่ (3.31)

$$H_n(S) = \frac{1}{(S+S_1)(S+S_2)...(S+S_k)...(S+S_n)} \quad ...(3.31)$$

โดยโพลของ TUT ที่ k คือ

$$S_k = |S_k|e^{-j\theta_k} \quad ...(3.32)$$

จากสมการ (3.32) ผลตอบสนองทางขนาด (magnitude) และ (phase) [15] เขียนได้ว่า

$$S_k = |S_{ku}|^{1-m} |S_{kt}|^m \quad ...(3.33)$$

โดยที่

$$\theta_k = \theta_{ku} - m(\theta_{ku} - \theta_{kt}) \quad ...(3.34)$$

$|S_{ku}|$ และ $\theta_{ku}$ เป็นขนาดและเฟสของอุลตร้าสเฟียริคัล โพลิโนเมียล ส่วน $|S_{kt}|$ และ $\theta_{kt}$ เป็นขนาดและเฟสของทอมป์สันโพลิโนเมียล ส่วน m เป็นพารามิเตอร์ ที่ปรับค่าได้ (Transition Factor) ซึ่งอยู่ระหว่าง $0 \le m \le 1$ จากสมการ (3.33) และ (3.34) ถ้า $m = 0$ จะได้ค่าโพลของอุลตร้าสเฟียริคัลอย่างเดียว

$$S_k = S_{ku} = |S_{ku}|e^{-j\theta_{ku}} \quad ...(3.35)$$

ในทำนองเดียวกันถ้า $m = 1$ จะเป็นค่าโพลของทอมป์สัน

$$S_k = S_{kt} = |S_{kt}|e^{-j\theta_{kt}} \quad ...(3.36)$$

รากของอุลตร้าสเฟียริคัลโพลิโนเมียล กำหนดให้ที่ $\alpha = 1$ และรากของทอมป์สันโพลิโนเมียลอันดับที่ n สามารถเขียนได้ดังตารางที่ 3.6

| รากของอุลตร้าสเฟียริคัล | | | | รากของทอมป์สัน | | |
|---|---|---|---|---|---|---|
| n | pole | $\|S_{ku}\|$ | $\theta_{ku}$ | pole | $\|S_{kt}\|$ | $\theta_{kt}$ |
| 1 | -1 | 1 | $0^{\circ}$ | -1 | 1 | $0^{\circ}$ |
| 2 | -0.7157±j0.8888 | 1 | $\pm 51^{\circ}$ | -1.500±j0.8665 | 1.7320 | $\pm 30^{\circ}$ |
| 3 | -0.6606 | .6606 | $0^{\circ}$ | -2.3221 | 2.3221 | $0^{\circ}$ |
| | -0.3303±j0.8694 | .9300 | $\pm 69.2^{\circ}$ | -1.8389±j1.7543 | 2.5414 | $\pm 43.65^{\circ}$ |
| 4 | -0.5322±j0.3773 | .6524 | $\pm 22.5^{\circ}$ | -2.8962±j0.8672 | 3.0232 | $\pm 16.67^{\circ}$ |
| | -0.2172±j0.9245 | .9497 | $\pm 67.5^{\circ}$ | -2.1037±j2.6574 | 3.3892 | $\pm 51.63^{\circ}$ |

ตารางที่ 3.6 แสดงรากของอุลตร้าสเฟียริคัลโพลิโนเมียลและทอมป์สันโพลิโนเมียล

เมื่อกำหนดให้ n = 4 และปรับค่า m อยู่ในระหว่าง 0 ถึง 1 จะได้รากของโพลิโนเมียล แบบ TUT ดังในตารางที่ 3.7

| m | ค่าราก |
|---|---|
| 0 | -0.5322 ± j0.3773 |
| | -0.2172 ± j0.9497 |
| 0.1 | -0.6345 ± j0.4195 |
| | -0.2925 ± j1.0391 |
| 0.2 | -0.7551 ± j0.4652 |
| | -0.3836 ± j1.1631 |
| 0.3 | -0.8975 ± j0.5125 |
| | -0.4953 ± j1.2998 |
| 0.4 | -1.0652 ± j0.5632 |
| | -0.6277 ± j1.4498 |
| 0.5 | -1.2623 ± j0.6157 |
| | -0.7853 ± j1.6131 |
| 0.6 | -1.4942 ± j0.6693 |
| | -0.9721 ± j1.7912 |
| 0.7 | -1.7661 ± j0.7232 |
| | -1.1931 ± j1.9831 |
| 0.8 | -2.0850 ± j0.7751 |
| | -1.4535 ± j2.1894 |
| 0.9 | -2.4589 ± j0.8246 |
| | -1.7594 ± j2.4110 |
| 1.0 | -2.8962 ± j0.8672 |
| | -2.1037 ± j2.6574 |

ตารางที่ 3.7 แสดงค่ารากของ TUT โพลิโนเมียล อันดับที่ 4

ในทำนองเดียวกัน โพลของ TBT ที่ k คือ

$$S_k = |S_k|e^{-j\theta_k} \quad ...(3.37)$$

จากสมการ (3.37) ผลตอบสนองทางขนาด (magnitude) และ (phase) [15] เขียนได้ว่า

$$S_k = |S_{kb}|^{1-m} |S_{kt}|^m \quad ...(3.38)$$

โดยที่

$$\theta_k = \theta_{kb} - m(\theta_{kb} - \theta_{kt}) \quad ...(3.39)$$

$|S_{kb}|$ และ $\theta_{kb}$ เป็นขนาดและเฟสของบัตเตอร์เวิร์ท โพลิโนเมียล ส่วน $|S_{kt}|$ และ $\theta_{kt}$ เป็นขนาดและเฟสของทอมป์สันโพลิโนเมียล ส่วน m เป็นพารามิเตอร์ ที่ปรับค่าได้ (Transition Factor)ซึ่งอยู่ระหว่าง $0 \le m \le 1$ จากสมการ (3.38) และ (3.39) ถ้า $m = 0$ จะได้ค่าโพลของบัตเตอรเวิร์ทอย่างเดียว

$$S_k = S_{kb} = |S_{kb}|e^{-j\theta_{kb}} \quad ...(3.40)$$

ในทำนองเดียวกันถ้า $m = 1$ จะเป็นค่าโพลของทอมป์สัน

$$S_k = S_{kt} = |S_{kt}|e^{-j\theta_{kt}} \quad ...(3.41)$$

รากของบัตเตอร์เวิร์ทโพลิโนเมียล และรากของทอมป์สันโพลิโนเมียลอันดับที่ 4 สามารถเขียนได้ดังตารางที่ 3.8

| รากของบัตเตอร์เวิร์ท | | | | รากของทอมป์สัน | | |
|---|---|---|---|---|---|---|
| n | pole | $\|S_{kb}\|$ | $\theta_{kb}$ | pole | $\|S_{kt}\|$ | $\theta_{kt}$ |
| 4 | -0.9239±j0.3827 | 1 | ±22.5° | -2.8962±j0.8672 | 3.0232 | ±16.67° |
| | -0.3827±j0.9239 | 1 | ±67.5° | -2.1037±j2.6574 | 3.3892 | ±51.63° |

ตารางที่ 3.8 แสดงรากของอุลตร้าสเฟียริคัลโพลิโนเมียลและทอมป์สันโพลิโนเมียล

เมื่อกำหนดให้ n = 4 และปรับค่า m อยู่ในระหว่าง 0 ~ 1 จะได้รากของโพลิโนเมียล แบบ TBT ดังในตารางที่ 3.9

| m | ค่าราก |
|---|---|
| 0 | -0.3827 ± j0.9239 |
| | -0.9239 ± j0.3827 |
| 0.1 | -0.4611 ± j1.0314 |
| | -1.0362 ± j0.4169 |
| 0.2 | -0.5530 ± j1.1504 |
| | -1.1621 ± j0.4539 |
| 0.3 | -0.6605 ± j1.2820 |
| | -1.3032 ± j0.4939 |
| 0.4 | -0.7861 ± j1.4272 |
| | -1.4611 ± j0.5366 |
| 0.5 | -0.9325 ± j1.5872 |
| | -1.6381 ± j0.5828 |
| 0.6 | -1.1028 ± j1.7634 |
| | -1.8363 ± j0.6323 |
| 0.7 | -1.3007 ± j1.9571 |
| | -2.0582 ± j0.6854 |
| 0.8 | -1.5303 ± j2.1696 |
| | -2.3066 ± j0.7421 |
| 0.9 | -1.7962 ± j2.4025 |
| | -2.5847 ± j0.8027 |
| 1.0 | -2.8962 ± j0.8672 |
| | -2.1037 ± j2.6574 |

ตารางที่ 3.9 แสดงค่ารากของ TBT โพลิโนเมียล อันดับที่ 4

## 3.4 การประมาณฟังก์ชั่นพัลซ์ซายน์กำลังสองด้วยทรานสฟิชชั่นนัลอุลตร้าสเฟียริคัล-ทอมสัน และบัตเตอร์เวิร์ท-ทอมป์สันโพลิโนเมียล

การประมาณค่าทรานส์เฟอร์ฟังก์ชั่น N(S)/D(S) เพื่อให้ได้ผลตอบสนองที่ดีที่สุด [12] ในช่วงของผลตอบสนองระหว่างอุลตร้าสเฟียริคัลกับทอมป์สันโพลิโนเมียล หรือ บัตเตอร์เวิร์ทกับทอมป์สันโพลิโนเมียลโดยวิธีการปรับค่าพารามิเตอร์ในส่วน D(S) แล้วจึงทำการประมาณค่าของ N(S) ด้วย Least square เพื่อหาค่าสัมประสิทธิ์ของตัวเศษ

กำหนดให้ทรานส์เฟอร์ฟังก์ชั่นที่ประมาณ เป็น

$$H(S) = \frac{N(S)}{D(S)} \qquad ...(3.42)$$

โดยที่ D(S) เป็น ทรานสฟิชชั่นนัลอุลตร้าสเฟียริคัล-ทอมสัน โพลิโนเมียล หรือ บัตเตอร์เวิร์ท-ทอมป์สันโพลิโนเมียล ฉะนั้นสมการ (3.42) เขียนใหม่ได้ว่า

$$H(S) = \frac{a_0+a_1S+.....+a_mS^m}{\prod_{v=1}^{M}(S-S_v)\prod_{\mu=1}^{N}(S-S_\mu)} = \frac{\sum_{i=0}^{n} a_iS^i}{\prod_{v=1}^{M}(S-S_v)\prod_{\mu=1}^{N}(S-S_\mu)} \qquad ...(3.43)$$

โดยที่ $S_v$ โพลของทรานส์เฟอร์ฟังก์ชั่น ซึ่งเป็น Complex

$S_\mu$ โพลของทรานส์เฟอร์ฟังก์ชั่น ซึ่งเป็น Real

$a_i$ เป็นสัมประสิทธิ์ของเศษที่ต้องการหา

จากสมการ (3.43) เขียนอยู่ในรูปของผลบ่ายของเวลาได้เป็น

$$h(t) = \sum_{v=1}^{M}[2A_ve^{-\alpha_v t}\cos\beta_v t + 2B_ve^{-\alpha_v t}\sin\beta_v t] + \sum_{\mu=1}^{N} c_\mu e^{-\sigma_\mu t} \qquad ...(3.44)$$

เมื่อ $v = 1,2,...,M;\ \mu = 1,2,...,N$

$\alpha_v, \beta_v$ = Complex Poles

$\sigma_\mu$ = Real Poles

แทนค่ารากของ TUT จากตารางที่ 3.6 ลงใน (3.44)

โดยเริ่มที่ m = 0

$$h(t) = (2A_1 e^{-0.5322t} \cos(0.3773t) + 2B_1 e^{-0.5322t} \sin(0.3773t)) + (2A_2 e^{-0.2172t} \cos(0.9497t) + 2B_2 e^{-0.2172t} \sin(0.9497t)) \quad ...(3.45)$$

และแทนค่ารากที่ m = 0.1 , 0.2 ,... 1 ตามลำดับ
ค่าของ Least mean square error ในช่วง 0 - 2T เขียนได้ว่า

$$E_2 = \int_0^{2T} [\, f(t) - h(t) \,]^2 \, dt \quad ...(3.46)$$

โดยที่ f(t) เป็นพัลซ์ซายน์กำลังสองและกำหนดให้ 2T = 1 วินาที
เพื่อให้ h(t) มีค่าผิดพลาดน้อยที่สุดนั่นคือ $\partial E_2/\partial A_v = 0, \partial E_2/\partial B_v = 0$ และ $\partial E_2/\partial C_\mu = 0$
เราจะได้สมการจำนวน 2M + N ตัว

$$\int_0^{2T} [\, f(t) - h(t) \,] e^{-\alpha_v t} \cos \beta_v t \, dt = 0$$

$$\int_0^{2T} [\, f(t) - h(t) \,] e^{-\alpha_v t} \sin \beta_v t \, dt = 0$$

$$\int_0^{2T} [\, f(t) - h(t) \,] e^{-\sigma_\mu t} \, dt = 0 \quad ...(3.47)$$

แต่ $\alpha_v, \beta_v$ และ $\sigma_\mu$ เป็นค่าคงที่ สมการ 2M+N เป็นสมการลิเนียร์ (Linear Equation) $A_v, A_v$ และ $C_\mu$ ไม่ทราบค่า สัมประสิทธิ์ของสมการสามารถหาค่าได้โดยการอินทิเกรต (Integration) สมการที่ 3.47 และสมการที่ 3.48 เป็นการอินทิเกรตในเทอมคงที่

$$\int_0^{2T} f(t) e^{-\alpha_v t} \cos \beta_v t \, dt = 0$$

$$\int_0^{2T} f(t) e^{-\alpha_v t} \sin \beta_v t \, dt = 0$$

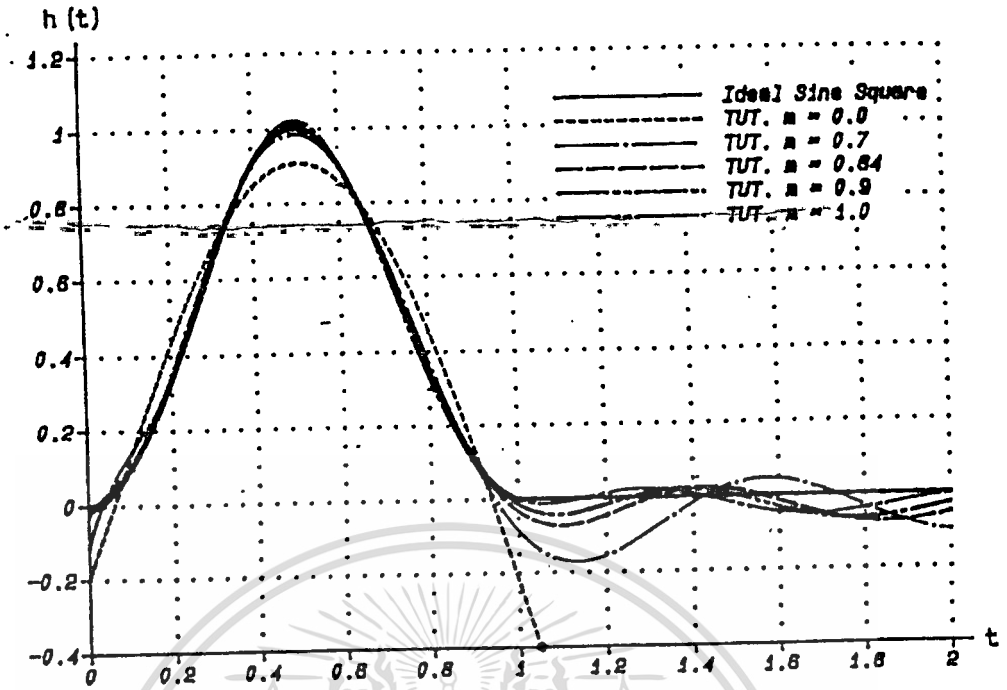
$$\int_0^{2T} f(t)\, e^{-\sigma_\mu t}\, dt = 0 \qquad \ldots(3.48)$$

สัมประสิทธิ์ $A_1$, $B_1$, $A_2$, $B_2$ หาได้จากการอินทิเกรตของสมการที่ (3.47) ที่ m = 0

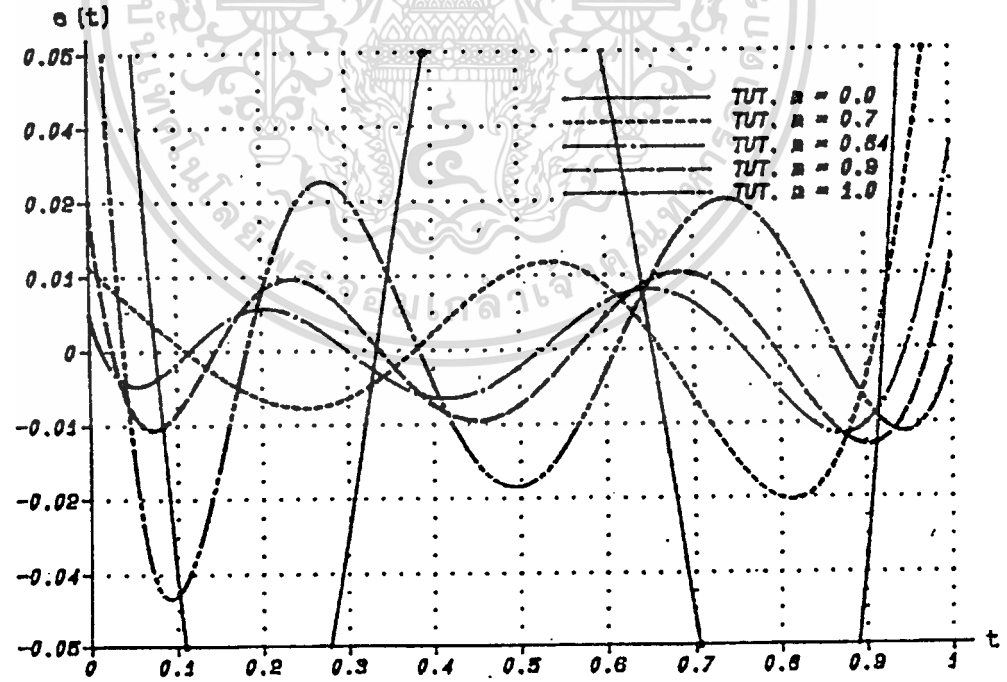
$$h(t) = 2(0.0232)e^{-0.5322t}\cos(0.3773t)+2(-0.6883)e^{-0.5322t}\sin(0.3773t) +2(-0.1195)e^{-0.2172t}\cos(0.9497t)+2(0.8374)e^{-0.2172t}\sin(0.9497t) \qquad \ldots(3.49)$$

และหา h(t) ที่ m = 0.1, 0.2, ....., 1 ตามลำดับ

จากฟังก์ชั่น h(t) ในสมการ (3.49) ที่ m ต่างๆ สามารถนำไปพล็อตในขอบข่ายของเวลาได้ดังรูปที่ 3.1 และรูปที่ 3.2




รูปที่ 3.1 แสดงค่าประมาณพัลซ์ซายน์กำลังสอง h(t) ของ TUT




รูปที่ 3.2 แสดงค่าผิดพลาดจากการประมาณฟังก์ชั่น h(t) ของ TUT

จากรูปที่ 3.1 และรูปที่ 3.2 จะเห็นว่า h(t) ที่ m = 0.7 ถึง m = 0.9 จะให้พัลซ์ซายน์กำลังสองใกล้เคียงกับอุดมคติ ซึ่งสามารถคำนวณหาค่าผิดพลาดได้ดังสมการที่ (3.49) และผลจากการคำนวณค่าผิดพลาดได้จากตารางที่ 3.10

$$E = \left| \int_0^1 [\, f(t) - h(t) \,]\, dt \right| \qquad \ldots\ldots(3.50)$$

โดยที่ f(t) เป็นฟังก์ชั่นพัลซ์ซายน์กำลังสองในอุดมคติ

h(t) เป็นฟังก์ชั่นที่ประมาณด้วย TUT.

E เป็นค่าผิดพลาดจากการประมาณฟังก์ชั่น

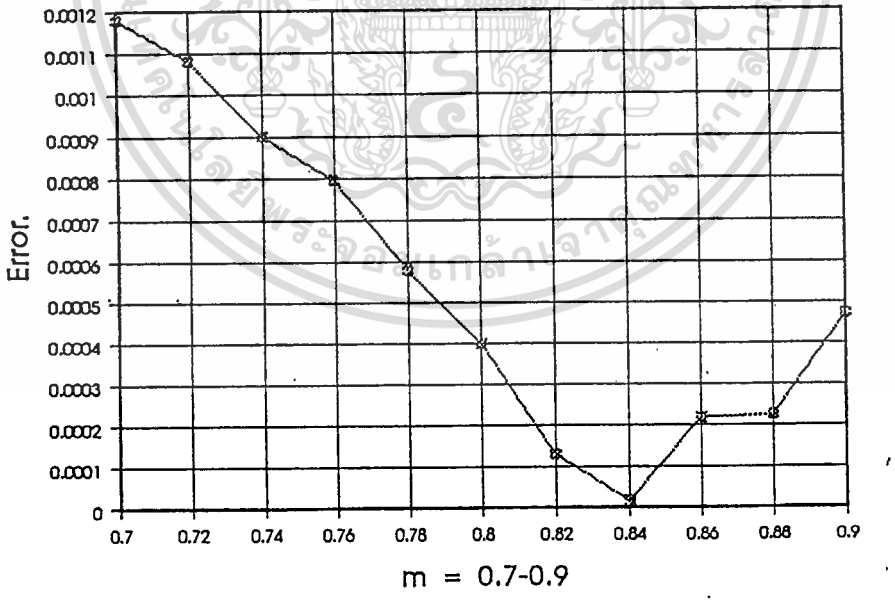
| m | Pole1 - Pole2 | Pole3 - Pole4 | Error. |
|---|---|---|---|
| 0.0 | -0.5322±j0.3773 | -0.2172±j0.9497 | 0.000,241,974 |
| 0.1 | -0.6345±j0.4195 | -0.2925±j1.0391 | 0.014,441,735 |
| 0.2 | -0.7551±j0.4652 | -0.3836±j1.1631 | 0.000,552,202 |
| 0.3 | -0.8975±j0.5152 | -0.4953±j1.2998 | 0.000,893,814 |
| 0.4 | -1.0652±j0.5632 | -0.6277±j1.4498 | 0.000,982,793 |
| 0.5 | -1.2623±j0.6157 | -0.7853±j1.6131 | 0.001,261,447 |
| 0.6 | -1.4942±j0.6693 | -0.9721±j1.7912 | 0.001,420,264 |
| 0.7 | -1.7661±j0.7232 | -1.1931±j1.9831 | 0.001,179,686 |
| 0.8 | -2.0850±j0.7751 | -1.4535±j2.1894 | 0.000,400,439 |
| 0.9 | -2.4589±j0.8246 | -1.7594±j2.4110 | 0.000,471,956 |
| 1.0 | -2.8962±j0.8672 | -2.1037±j2.6574 | 0.001,505,362 |

ตารางที่ 3.10 แสดงการคำนวณค่าผิดพลาดของ TUT โดยการปรับค่า m = 0 ถึง m = 1

จากรูปที่ 3.1 รูปที่ 3.2 และตารางการคำนวณค่าผิดพลาด สามารถปรับค่า m ตั้งแต่ 0.7 ถึง 0.9 ทีละ 0.02 step ค่าความผิดพลาดสามารถคำนวณและพล๊อตเป็นกราฟได้ดังตารางที่ 3.11 และกราฟที่ 3.1

| m | Pole1 - Pole2 | Pole3 - Pole4 | Error. |
|---|---|---|---|
| 0.07 | -1.7661±j0.7232 | -1.1931±j1.9831 | 0.001,179,686 |
| 0.72 | -1.8261±j0.7337 | -1.2413±j2.0233 | 0.001,080,632 |
| 0.74 | -1.8870±j0.7444 | -1.2921±j2.0637 | 0.000,900,100 |
| 0.76 | -1.9515±j0.7549 | -1.3441±j2.1049 | 0.000,796,044 |
| 0.78 | -2.0172±j0.7655 | -1.3979±j2.1465 | 0.000,578,949 |
| 0.80 | -2.0850±j0.7751 | -1.4535±j2.1894 | 0.000,400,439 |
| 0.82 | -2.1551±j0.7571 | -1.4891±j2.2309 | 0.000,130,099 |
| 0.84 | -2.2302±j0.7961 | -1.5701±j2.2760 | 0.000,016,042 |
| 0.86 | -2.3021±j0.8051 | -1.6322±j2.3210 | 0.000,219,026 |
| 0.88 | -2.3791±j0.8151 | -1.6541±j2.3103 | 0.000,226,756 |
| 0.90 | -2.4589±j0.8246 | -1.7594±j2.4110 | 0.000,471,956 |

ตารางที่ 3.11 แสดงการคำนวณค่าผิดพลาดของ TUT โดยการปรับค่า m = 0.7 ถึง m = 0.9




กราฟที่ 3.1 แสดงค่าผิดพลาด E ของ TUT โดยการปรับค่า m ตั้งแต่ 0.7 ถึง 0.9

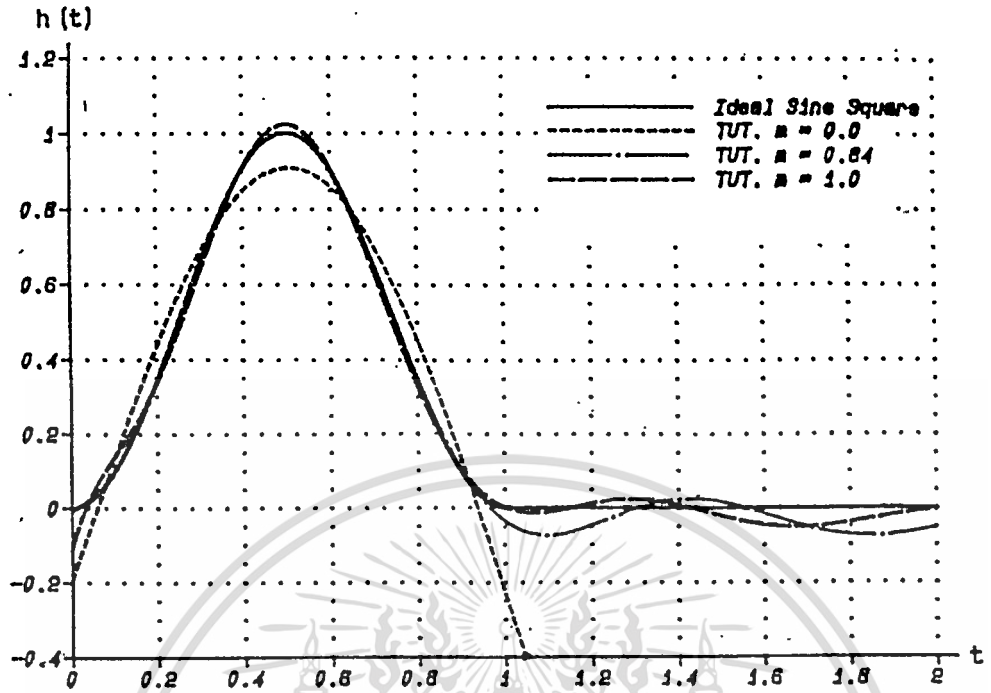
จากรูปที่ 3.2 และกราฟที่ 3.1 เป็นการพล็อตค่าผิดพลาด ซึ่งจะได้ว่าค่าฟังก์ชั่น h(t) ที่ m = 0.84 นั้น จะให้ค่าผิดพลาดต่ำที่สุดในการประมาณพัลซ์ซายน์กำลังสอง ด้วยวิธี ทรานส์ซิทชั่นนัล อุลตร้าสเฟียริคัล ทอมป์สัน โพลิโนเมียล

เมื่อ m = 0.84 ค่า h(t) สามารถเขียนได้ว่า

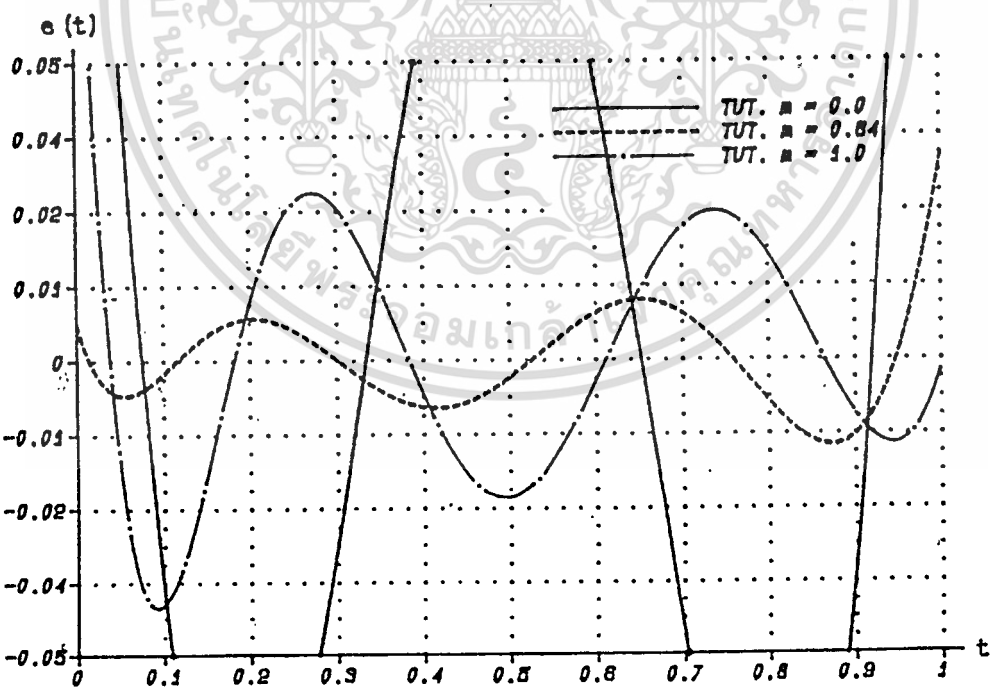
$$h(t) = 0.3130e^{-2.2302t}\cos(0.7961t)+2.2678e^{-2.2302t}\sin(0.7961t) -0.3192e^{-1.5701t}\cos(2.2769t)-0.6926e^{-1.5701t}\sin(2.2769t) \quad ...(3.51)$$

จากสมการ (3.51) ทำการ Take Laplace จะได้ทรานส์เฟอร์ฟังก์ชั่นดัง

$$H(S) = \frac{-0.0062S^3 - 0.0156S^2 -0.8037S + 7.4968}{S^4 + 7.6006S^3 + 27.2636S^2 + 51.7287S + 42.8953} \quad ...(3.52)$$




รูปที่ 3.3 แสดงค่าประมาณพัลซ์ซายน์กำลังสอง h(t) ของ TUT




รูปที่ 3.4 แสดงค่าผิดพลาดจากการประมาณฟังก์ชั่น h(t) ของ TUT

ในทำนองเดียวกัน แทนค่ารากของ TBT จากตารางที่ 3.9 ลงใน (3.44)
โดยเริ่มที่ m = 0

$$h(t) = (2A_1\, e^{-0.3827t}\cos(0.9239t)+2B_1\, e^{-0.3827t}\sin(0.9239t)) +(2A_2\, e^{-0.9239t}\cos(0.3827t)+2B_2\, e^{-0.9239t}\sin(0.3827t)) \quad ...(3.52)$$

และแทนค่ารากที่ m = 0.1 , 0.2 ,... 1 ตามลำดับ
ค่าของ Least mean square error ในช่วง 0 - 2T เขียนได้ว่า

$$E_2 = \int_0^{2T} [\, f(t) - h(t)\, ]^2\, dt \quad ...(3.53)$$

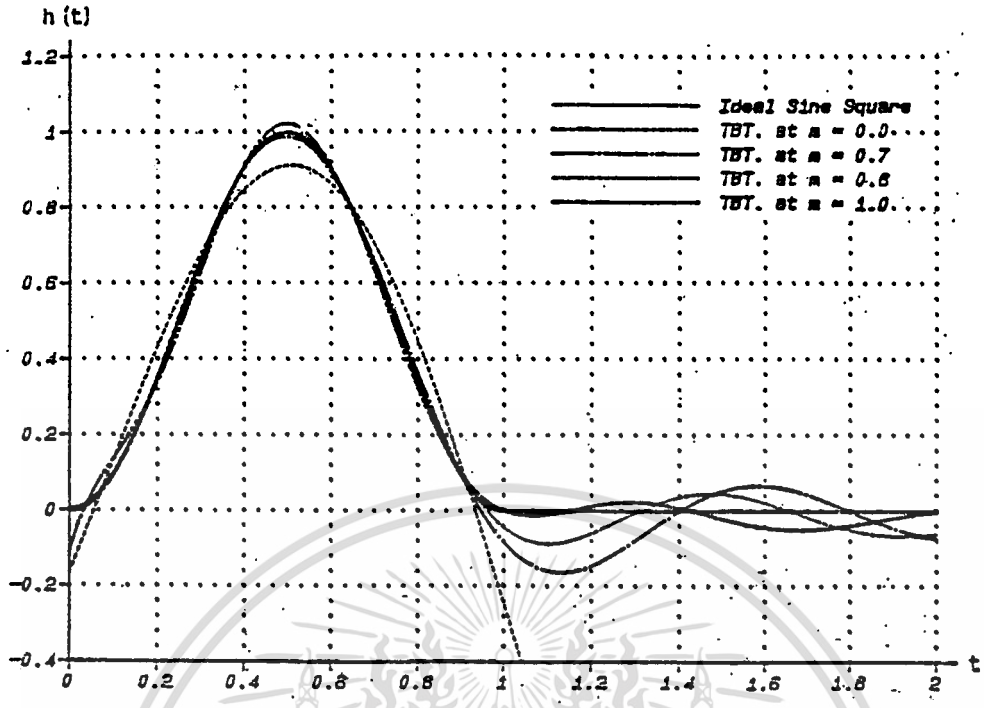
โดยที่ f(t) เป็นพัลซ์ซายน์กำลังสองและกำหนดให้ 2T = 1 วินาที

สัมประสิทธิ์ $A_1$, $B_1$, $A_2$, $B_2$ หาได้จากการอินทิเกรตของสมการที่ (3.47) ที่ m = 0

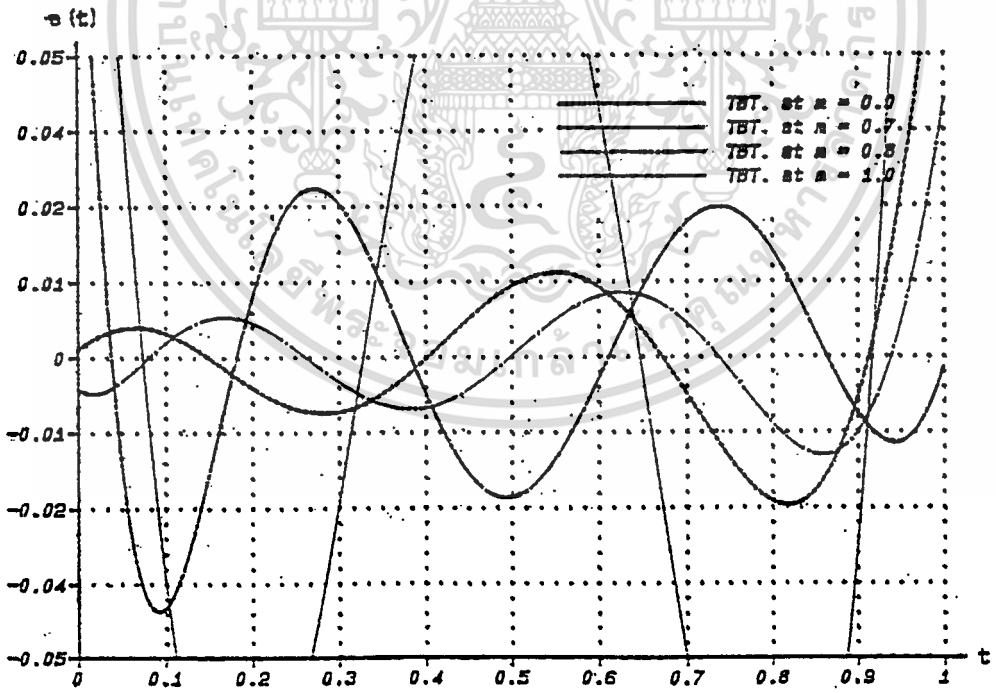
$$h(t) = 2(-0.2483)e^{-0.3827t}\cos(0.9239t)+2(1.0400)e^{-0.3827t}\sin(0.9239t) +2(0.1649)e^{-0.9239t}\cos(0.3827t)+2(-1.2863)e^{-0.9239t}\sin(0.3827t) \quad ...(3.54)$$

และหา h(t) ที่ m = 0.1, 0.2, ....., 1 ตามลำดับ

จากฟังก์ชั่น h(t) ในสมการ (3.54) ที่ m ต่างๆ สามารถนำไปพล็อตในชอบข่ายของเวลา ได้ดังรูปที่ 3.5 และรูปที่ 3.6




รูปที่ 3.5 แสดงค่าประมาณพัลซ์ซายน์กำลังสอง h(t) ของ TBT




รูปที่ 3.6 แสดงค่าผิดพลาดจากการประมาณฟังก์ชั่น h(t) ของ TBT

| m | Pole1 - Pole2 | Pole3 - Pole4 | Error. |
|---|---|---|---|
| 0.0 | -0.3827±j0.9239 | -0.9239±j0.3827 | 0.000,525,179 |
| 0.1 | -0.4611±j1.0314 | -1.0362±j0.4169 | 0.000,390,579 |
| 0.2 | -0.5530±j1.1504 | -0.4764±j0.6150 | 0.000,567,865 |
| 0.3 | -0.6605±j1.2820 | -1.3032±j0.4936 | 0.000,742,589 |
| 0.4 | -0.7861±j1.4272 | -1.4611±j0.5366 | 0.001,774,810 |
| 0.5 | -0.9325±j1.5872 | -1.6381±j0.5828 | 0.001,145,617 |
| 0.6 | -1.1028±j1.7634 | -1.8363±j0.6323 | 0.001,232,541 |
| 0.7 | -1.3007±j1.9571 | -2.0582±j0.6854 | 0.000,871,978 |
| 0.8 | -1.5303±j2.1696 | -2.3066±j0.7421 | 0.000,028,955 |
| 0.9 | -1.7962±j2.4025 | -2.5847±j0.8027 | 0.000,706,178 |
| 1.0 | -2.1037±j2.6572 | -2.8962±j0.8672 | 0.001,494,359 |

ตารางที่ 3.12 แสดงการคำนวณค่าผิดพลาดของ TBT โดยการปรับค่า m = 0 ถึง m = 1

จากรูปที่ 3.5 รูปที่ 3.6 และตารางการคำนวณค่าผิดพลาด สามารถปรับค่า m ตั้งแต่ 0.7 ถึง 0.9 ทีละ 0.02 step ค่าความผิดพลาดสามารถคำนวณได้ดังตารางที่ 3.13

| m | Pole1 - Pole2 | Pole3 - Pole4 | Error. |
|---|---|---|---|
| 0.07 | -1.3007±j1.9572 | -2.0582±j0.6854 | 0.000,871,978 |
| 0.72 | -1.3441±j1.9980 | -2.1057±j0.6964 | 0.000,728,632 |
| 0.74 | -1.3884±j2.0399 | -2.1542±j0.7078 | 0.010,917,183 |
| 0.76 | -1.4344±j2.0824 | -2.2038±j0.7191 | 0.000,455,149 |
| 0.78 | -1.4817±j2.1256 | -2.2547±j0.7304 | 0.000,393,707 |
| 0.80 | -1.5303±j2.1696 | -2.3066±j0.7421 | 0.000,028,955 |
| 0.82 | -1.5803±j2.2147 | -2.3598±j0.7540 | 0.000,143,585 |
| 0.84 | -1.6320±j2.2603 | -2.4142±j0.7658 | 0.000,357,270 |
| 0.86 | -1.6852±j2.3068 | -2.4697±j0.7782 | 0.000,505,375 |
| 0.88 | -1.7401±j2.3541 | -2.5266±j0.7903 | 0.000,696,772 |
| 0.90 | -1.7962±j2.4025 | -2.5847±j0.8027 | 0.000,706,178 |

ตารางที่ 3.13 แสดงการคำนวณค่าผิดพลาดของ TBT โดยการปรับค่า m = 0.7 ถึง m = 0.9

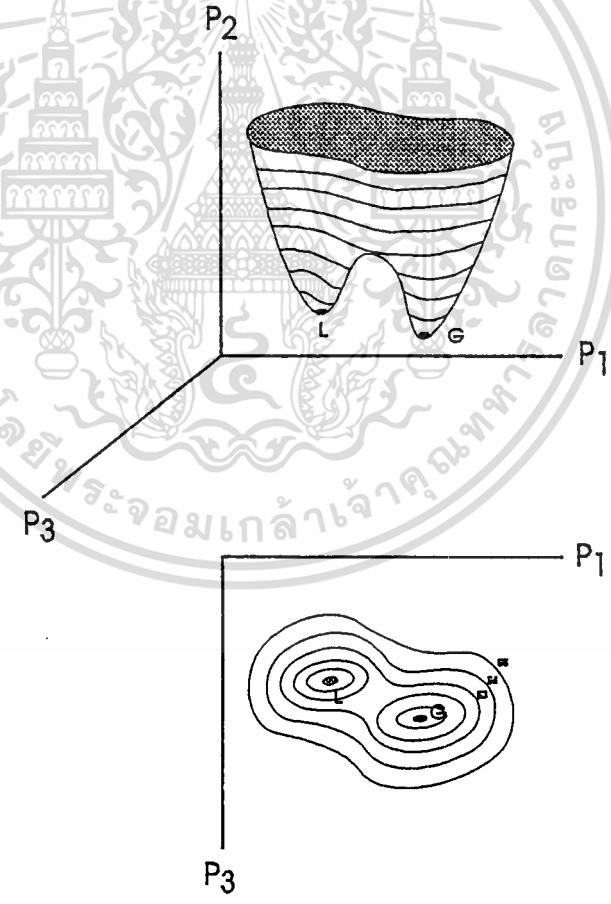
จากตารางที่ 3.11 และตารางที่ 3.12 เป็นการคำนวณค่าผิดพลาด ซึ่งจะได้ว่าค่าฟังก์ชั่น $h(t)$ ที่ $m = 0.84$ ของ TUT นั้น จะให้ค่าผิดพลาดต่ำที่สุดในการประมาณพัลซ์ซายน์กำลังสอง ด้วยวิธี ทรานส์ชิทชั่นนัล อุลตร้าสเฟียริคัล-ทอมป์สัน โพลิโนเมียล

ค่าสัมประสิทธิ์ของฟังก์ชั่น TUT ในสมการ (3.50) จะเป็นตัวกำหนดค่าสัมประสิทธิ์เริ่มต้นตามกรรมวิธี Optimization ในโดเมนเวลาโดยวิธีการของ เน็คกาทีฟ-เกรเดียนท์ เพื่อให้ได้ผลตอบสนองใกล้เคียงกับพัลซ์ซายน์กำลังสองในทางอุดมคติ

# บทที่ 4

## การประมาณฟังก์ชั่นด้วยกรรมวิธี Optimization

ในการประมาณฟังก์ชั่นด้วยกรรมวิธี Optimization ก่อนอื่นจะต้องกำหนดรายละเอียดของฟังก์ชั่นที่ต้องการประมาณ (Approximation Function) กับฟังก์ชั่นในทางอุดมคติ ฟังก์ชั่นที่ประมาณจะต้องมีค่าสัมประสิทธิ์เริ่มต้นเพื่อใช้ในการปรับปรุงค่าสัมประสิทธิ์ใหม่ ด้วยกรรมวิธี Optimization ซึ่งการปรับปรุงฟังก์ชั่นที่ประมาณให้ได้ฟังก์ชั่นใกล้เคียงกับฟังก์ชั่นในทางอุดมคติ ทุกๆครั้งในการปรับปรุงสัมประสิทธิ์ใหม่ของฟังก์ชั่นจะใช้ Optimality Index ($E^j$) (Error Function or Scalar Function) [20] ตัวบ่งชี้ถึงคุณภาพของผลตอบสนองที่ได้รับจากการปรับปรุงสัมประสิทธิ์ใหม่ ($P_i^j$) หากพิจารณาการเปลี่ยนแปลงของ $E^{(j)}$ และค่าสัมประสิทธิ์ $P_i^j$ ซึ่งสามารถแสดงเส้นทาง Contour ของการเปลี่ยนแปลงได้ดังรูปที่ 4.1




รูป 4.1 แสดง ก) Error Surface ข) เส้นทาง Contour

จากรูป 4.1 ในการปรับปรุงสัมประสิทธิ์ในแต่ละครั้งของกรรมวิธี Optimization สามารถลดระดับค่าผิดพลาดของ $E_5$ มายัง $E_4$ หรือ $E_4$ มายัง $E_3$ เป็นต้น และจากรูป 4.1 ได้แสดงจุด Optimum ที่จุด G มักจะเรียกว่า Global Optimum และอีกจุดหนึ่งที่ L มักเรียกว่า Local Optimum ซึ่งโดยปกติจุด Local Optimum จะมีมากมายใน Error Surface ซึ่งขึ้นอยู่กับการกำหนดค่าสัมประสิทธิ์เริ่มต้นให้เหมาะสม หากกำหนดค่าสัมประสิทธิ์เริ่มต้นที่ไม่เหมาะสม อาจต้องใช้เวลานานและอาจทำให้กรรมวิธี Optimization ล้มเหลวไป เพื่อหลีกเลี่ยงปัญหาดังกล่าว จึงกำหนดค่าสัมประสิทธิ์เริ่มต้นโดยใช้ ทรานส์ซิทชั่นนัลอุลตร้าสเฟียริคัล-ทอมป์สัน โพลิโนเมียล

## 4.1 กรรมวิธี Optimization

การคำนวณทางคณิตศาสตร์สำหรับการออกแบบเพื่อหาจุด Optimum นั้น ก่อนอื่นต้องกำหนดค่าจุดเริ่มต้น จากนั้นจึงคำนวณหาจุดที่ออกแบบต่อไป เมื่อได้จุดใหม่ที่ออกแบบแล้วจึงนำจุดดังกล่าวมาคำนวณซ้ำ (Iterative) จนกระทั่งได้จุดที่ดีที่สุดสำหรับการออกแบบหรือเรียกว่าจุด Optimum อัลกอลิทึมของการคำนวณแสดงรายละเอียดตามจำนวนครั้งของการคำนวณซ้ำ (Iterative) ได้

Vector Form $$p^{(j+1)} = p^{(j)} + \Delta p^{(j)} \qquad ...(4.1)$$

และ

Component Form $$p_i^{(j+1)} = p_i^{(j)} + \Delta p_i^{(j)} \qquad ...(4.2)$$

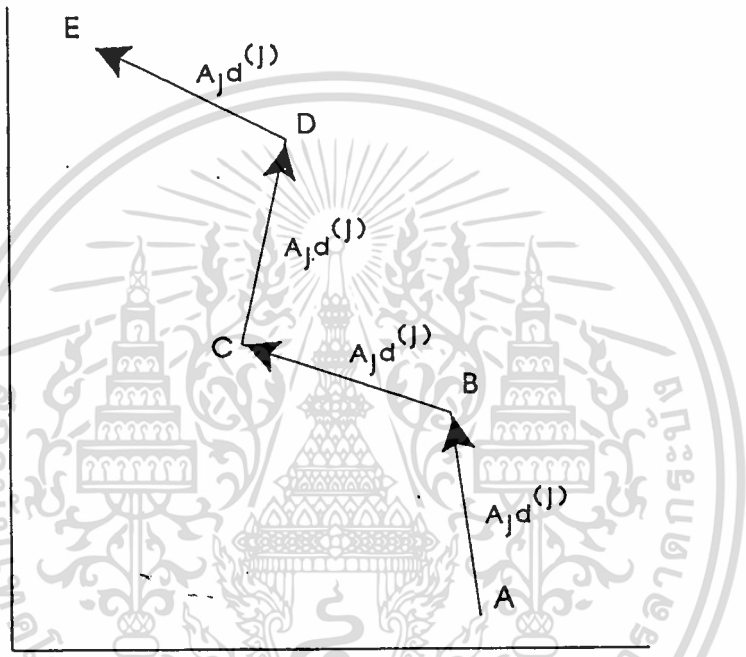
เมื่อ $j = 0,1,2...$

$i = 1,2,3...,n$

จากสมการ (4.1)และ(4.2) j แทนด้วยจำนวนครั้งของการคำนวนซ้ำ (Iteration) i เป็นจำนวนของตัวแปร (Variable) จุดเริ่มต้นของการคำนวณเท่ากับ $p^{(0)}$ และ $\Delta p^{(j)}$ แทนด้วยการเปลี่ยนแปลงจำนวนเล็กๆที่ Current Design การ Iterative จะกระทำอย่างต่อเนื่องจนกระทั่งได้จุด Optimum หรือการคำนวณได้จุดที่ยอมรับได้ ก็จะสิ้นสุดการคำนวณ การคำนวณหา $\Delta p^{(j)}$ขึ้นอยู่กับค่าสเกลล่าฟังก์ชั่น (Scalar Function) และค่าอนุพันธ์ (Derivative) ของ Scalar Function ที่ตำแหน่งจุดออกแบบ การคำนวณหา $\Delta p^{(j)}$ หาได้จากสมการ (4.3)

$$\Delta p^{(j)} = A_j d^{(j)} \qquad ...(4.3)$$

เมื่อ $d^{(j)}$ เป็นทิศทางที่คำนวณได้จากการเคลื่อนที่ใน Design Space และ $A_j$ เป็นค่า สเกลล่า เฟกเตอร์ (Scalar Factor) ที่มีค่าเป็นบวกซึ่งมีทิศทางเดียวกันกับทิศทางที่คำนวณได้จากการเคลื่อนที่ใน Design Space หรือเรียกว่า Step Size ในทิศทางเดียวกัน การคำนวณหาค่า $\Delta p^{(j)}$ จะแยกการคำนวณคือ การคำนวณหาทิศทางการเคลื่อนที่ และการคำนวณหาความยาวของ Step Size ตามทิศทางเดียวกัน การคำนวณจะเริ่มจากจุดแรกที่กำหนดไปยังจุดถัดไปดังแสดงในรูป 4.2




รูป 4.2 แสดงการ Iterative ของกรรมวิธี Optimization

จากรูป 4.2 จุด A เป็นจุด Starting Design จุด B เป็นจุด Current Design $d^{(j)}$ เป็นทิศทางที่คำนวณได้ และ $A_j$ เป็น Step Size เมื่อ $A_j d^{(j)}$ เป็นตัวบวกที่ Current Design เพราะฉะนั้นจะได้จุด C ใน Design Space และจากจุด C ก็จะคำนวณหาจุดอื่นต่อไป การคำนวณจะกระทำซ้ำต่อไปเรื่อยๆจนกระทั่งได้จุด Optimum

### 4.1.1 Descent Step Function

การคำนวณของ Iterative สำหรับกรรมวิธี Optimization เพื่อจะทำให้ได้จุด Optimum หรือ Scalar Function น้อยที่สุด กำหนดให้ฟังก์ชั่น f(p) เป็น Scalar Function ถ้า $p^{(j)}$ ไม่เป็นจุดที่ทำให้ฟังก์ชั่น f(p) ต่ำที่สุดจึงสามารถหาจุด $p^{(j+1)}$ ที่ทำให้ Scalar Function ของฟังก์ชั่น f(p) มีค่าน้อยกว่า Scalar Function ของฟังก์ชั่น f(p) ที่จุด $p^{(j)}$ ได้ดังสมการ

$$f(p^{(j+1)}) \quad < \quad f(p^{(j)}) \qquad ...(4.4)$$

ที่จุด $p^{(j+1)} = p^{(j)} + A_j d^{(j)}$ แทนลงในสมการที่ (4.4)

$$f(p^{(j)} + A_j d^{(j)}) \quad < \quad f(p^{(j)}) \qquad ...(4.5)$$

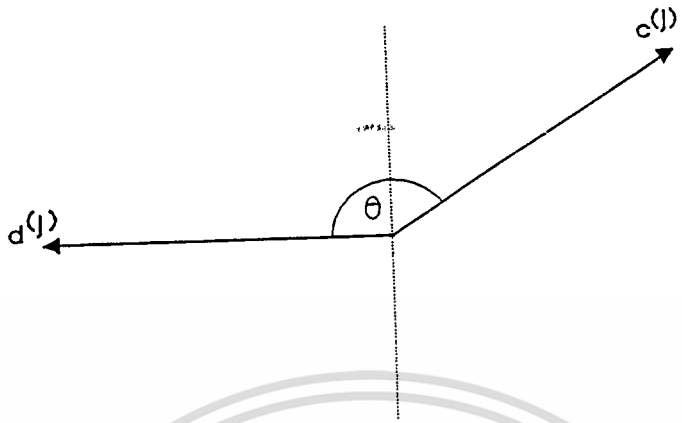
จากซีกซ้ายของสมการที่ (4.5) กระจายอนุกรมเทเลอย์ (Taylor Series) ที่จุด $p^{(j)}$ ซึ่งเป็นการ Approximating จะได้

$$f(p^{(j)}) + A_j(c^{(j)}.d^{(j)}) \quad < \quad f(p^{(j)}) \qquad ...(4.6)$$

เมื่อ $c^{(j)}$ เป็นเกรเดียนท์ (Gradient) ของฟังก์ชั่น f(p) ที่จุด $p^{(j)}$, $c^{(j)} = \nabla f(p^{(j)})$ , $d^{(j)}$ เป็นทิศทางที่คำนวณได้ และ (·) แทนด้วย Dot Product ของเวคเตอร์ (Vector) ทั้งสอง แต่ทางซีกซ้ายของสมการที่ (4.6) จะต้องมีค่าน้อยกว่าสมการทางซีกขวา เพราะฉะนั้นจะได้ว่า

$$A_j(c^{(j)}.d^{(j)}) \quad < \quad 0 \qquad ...(4.7)$$

เนื่องจาก $A_j$ เป็นสเกลล่าซึ่งมีค่าบวก ($A_j>0$) ,$c^{(j)}$ เป็นเกรเดียนท์ของ Scalar Function $d^{(j)}$ เป็นทิศทางที่คำนวณได้จุดดีที่สุด มุมระหว่างเวคเตอร์ $c^{(j)}$ และ $d^{(j)}$ จะต้องอยู่ระหว่าง 90 องศา ถึง 270 องศา($90 < \theta < 270$)ดังรูปที่ 4.2 ทุกๆครั้งของ Iterative จะได้ทิศทางที่ทำให้ Scalar Function ลดลง ทิศทางที่ทำให้ Scalar Function ลดลงสูงสุด (Maximun Scalar Function) จะต้องเป็นทิศทางของ เน็คกาทีฟ-เกรเดียนท์ (Negative Gradient) ที่จุด p ทุกๆจุดเล็กๆที่เคลื่อนไปตามทิศทาง เน็คกาทีฟ-เกรเดียนท์นี้ผลจะทำให้ Maximum Level Rate ของ Scalar Function ลดลงต่ำสุด เวคเตอร์ของเน็คกาทีฟ-เกรเดียนท์นั้นแทนด้วยทิศทางของ Steepest Descent สำหรับ Scalar Function



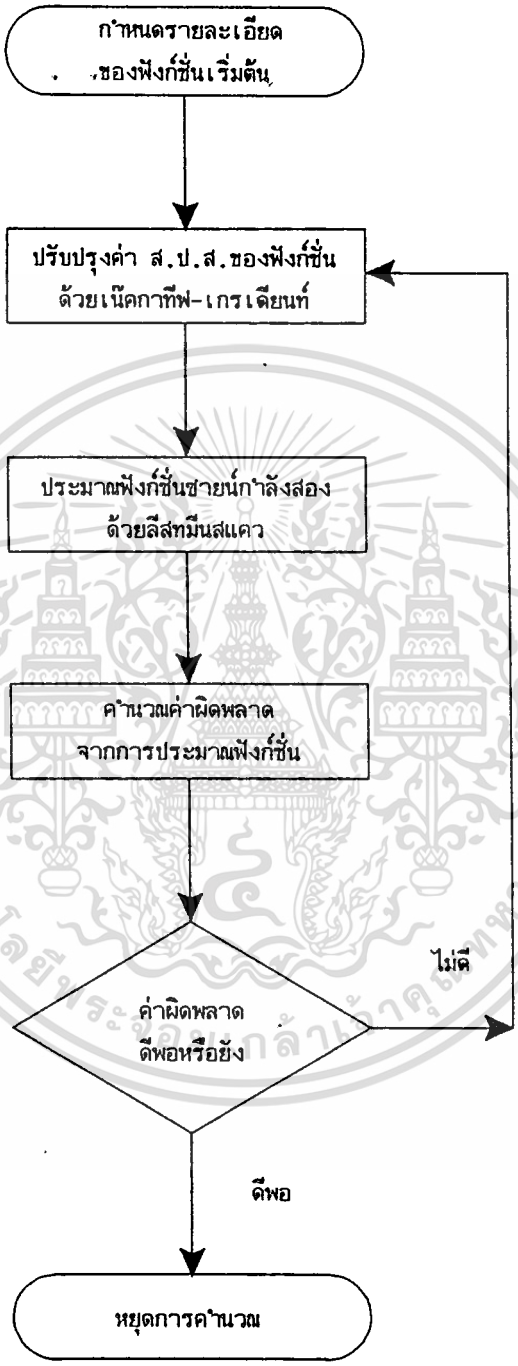

รูปที่ 4.3 แสดงทิศทาง $c^{(j)}$ และ $d^{(j)}$ ของ Scalar Function

4.1.2 Rate Convergence

ในการคำนวณแต่ละครั้งของ Iterative จนกระทั่งได้จุด Optimum ที่ทำให้ Scalar Function ดีที่สุด เพราะฉะนั้นเป็นสิ่งสำคัญที่จะต้องทำให้วิธีการคำนวณแต่ละครั้งทำให้อัตราการเปลี่ยนแปลงของ Scalar Function ที่สูงๆ (Maximum Rate Convergence) โดยปกติ Rate Convergence จะวัดได้โดยดูจำนวนครั้งของ Iterative และค่าของ Scalar Function ที่ คำนวณได้ ถ้าใช้จำนวนครั้งของ Iterative น้อยในการหาจุด Optimum แสดงว่า Rate Convergence ของวิธีการคำนวณของ Iterative Optimization ดีและลดเวลาในการคำนวณแต่ละครั้ง

## 4.2 การประมาณฟังก์ชั่นด้วยวิธี เน็คกาทีฟ-เกรเดียนท์

จากคุณสมบัติของ เกรเดียนท์-เวคเตอร์ สามารถจะกำหนดอัลกอลิทึมของ Iterative สำหรับกรรมวิธี Optimization ซึ่งใช้ในการประมาณฟังก์ชั่นด้วยวิธีเน็คกาทีฟ-เกรเดียนท์ซึ่งทำการปรับปรุงสัมประสิทธิ์ของฟังก์ชั่น เพื่อให้ค่าผิดพลาดจากการประมาณฟังก์ชั่นลดลงอย่างรวดเร็วและสามารถค้นหาจุดที่ดีที่สุด (Optimum) กรรมวิธี Optimization ประกอบเป็นขั้นตอนดังรูปที่ 4.4




รูปที่ 4.4 แสดงโฟว์ชาร์ตของกรรมวิธี Optimization

ขั้นตอนของการคำนวณแต่ละครั้งของการปรับปรุงสัมประสิทธิ์ ($p_i^{(j+1)}$) เขียนได้ว่า

$$P_i^{(j+1)} = P_i^{(j)} + \Delta P_i^{(j)} \qquad ...(4.8)$$

$j$ = จำนวนครั้งของการคำนวณซ้ำ (Iterative)

$i$ = จำนวนสัมประสิทธิ์ที่ปรับปรุง

$P^{(j+1)}$ = เป็นสัมประสิทธิ์ที่ปรับปรุงใหม่

$P_i^{(j)}$ = เป็นสัมประสิทธิ์ที่ Current Design

$\Delta P_i^{(j)}$ = เป็นการเปลี่ยนแปลงของสัมประสิทธิ์ที่ Current Design

จากสมการที่ (4.8) ค่าการเปลี่ยนแปลงของสัมประสิทธิ์ที่ Current Design $\Delta P_i^{(j)}$ หาได้จากสมการที่ (4.9)

$$\Delta P_i^{(j)} = kd^{(j)} \qquad ...(4.9)$$

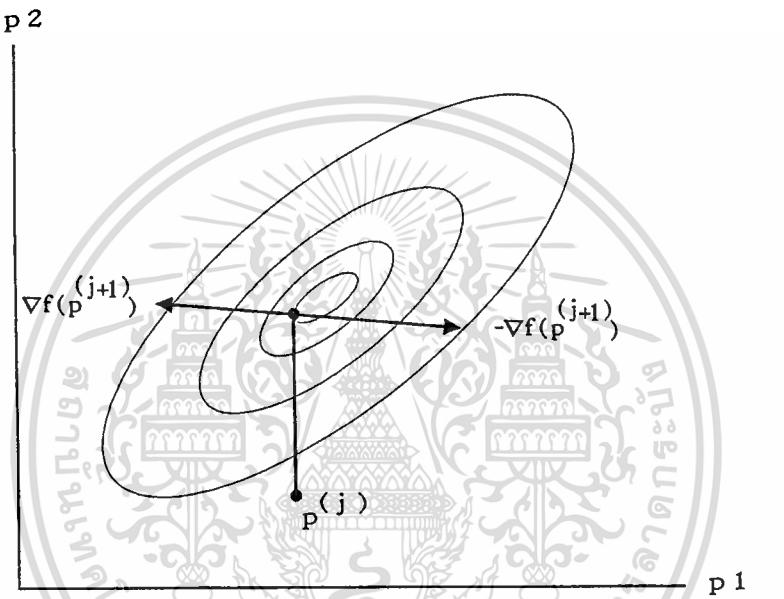
$k$ เป็นอัตราการเปลี่ยนแปลงสำพัทธ์ในสัมประสิทธิ์ $P_i^{(j)}$

$d^{(j)}$ เป็นทิศทางการเคลื่อนที่ๆจุดออกแบบ

จากสมการที่ (4.9) ค่า $k$ สามารถคำนวณได้จาก อัตราการเปลี่ยนแปลงในสัมประสิทธิ์ $P_i^{(j)}$ ดัง

$$k = \frac{\Delta max}{\left| \frac{\partial E^{(j)}}{\partial P_i^{(j)}} / P_i^{(j)} \right|} \qquad ...(4.10)$$

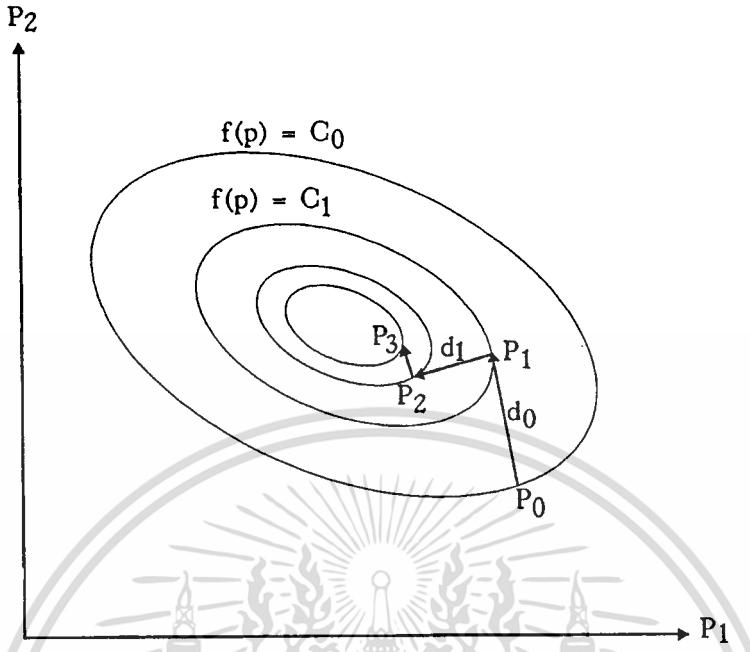
โดยที่ $\Delta max$ เป็นอัตราการเปลี่ยนแปลงสำพัทธ์ในสัมประสิทธิ์ $P_i^{(j)}$ ซึ่งจะมีค่าอยู่ระหว่าง 0 ถึง 50 เปอร์เซนต์ สำหรับการคำนวณหาทิศทางของ $d^{(j)}$ ที่ทำให้ค่าผิดพลาดจากการประมาณฟังก์ชั่นลดลงอย่างรวดเร็ว มีหลายวิธีด้วยกัน แต่วิธีที่ใช้ทิศทางของเน็คกาทีฟ-เกรเดียนท์ในการคำนวณของกรรมวิธี Optimization ดังรูปที่ 4.4 และยกตัวอย่างที่ใช้ในที่นี้มี 2 วิธี คือวิธีสตีปเปส-ดีเซน(Steepest Descent Method) และวิธีคอนจูเกจเกรเดียนท์ (Conjugate Gradient Method)




รูปที่ 4.5 แสดงทิศทางของเน็คกาทีฟ-เกรเดียนท์ที่จุด $p^{(j)}$

4.2.1 Steepest Descent Method

จากคุณสมบัติของ เน็คกาทีฟ-เวคเตอร์ ทิศทางที่ทำให้ค่าผิดพลาดจากการประมาณฟังก์ชั่น $E^{(j)}$ ลดลงอย่างรวดเร็วจะต้องเป็นทิศทาง เน็คกาทีฟ-เกรเดียนท์ ทุกๆครั้งของการปรับปรุงสัมประสิทธิ์ $P_i^{(j)}$ ของฟังก์ชั่น $E^{(j)}$ ไปตามทิศทาง เน็คกาทีฟ-เกรเดียนท์ ผลจะทำให้ค่าผิดพลาดของฟังก์ชั่น $E^{(j)}$ ลดลงอย่างรวดเร็ว เวคเตอร์ของเน็คกาทีฟ-เกรเดียนท์นั้นแทนด้วยทิศทางสตีปเปส-ดีเซน $d^{(j)}$ สำหรับฟังก์ชั่น $E^{(j)}$ ที่จุด $p^{(j)}$ แต่ละครั้งของการปรับปรุงสัมประสิทธิ์ $p_i^{(j)}$ ดังแสดงในรูปที่ 4.5




รูปที่ 4.6 แสดงทิศทาง Iterative ของวิธี สตีปเปส-ดีเซน

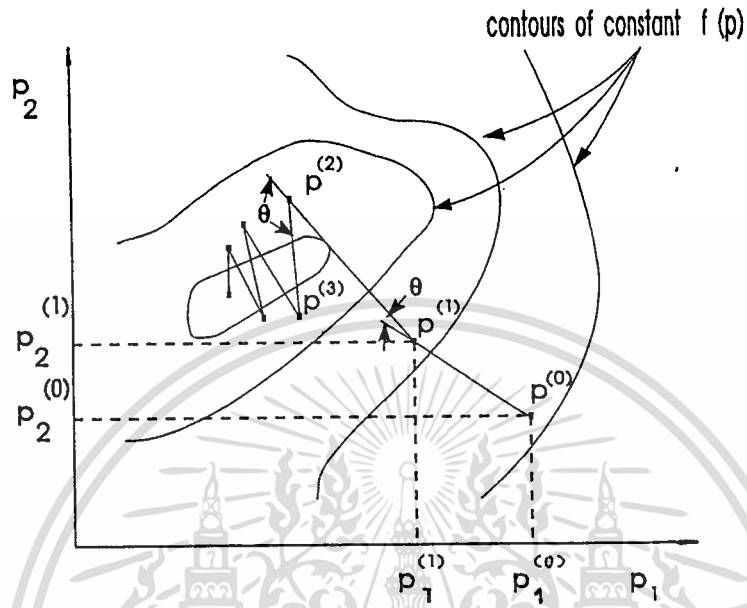
ทิศทางของวิธี สตีปเปส-ดีเซน $d^{(j)}$ ที่จุด $p^{(j)}$ คำนวณได้ดัง

$$d^{(j)} = -\left\{ \frac{\partial E^{(j)}}{\partial P_i^{(j)}} \right\} \qquad ...(4.11)$$

จากรูปที่ 4.6 จะเริ่มการออกแบบที่จุด Starting Design ($P^{(0)}$) ทิศทาง สตีปเปส-ดีเซน หาได้จากการคำนวณของ เน็คกาทีฟ-เกรเดียนท์ ที่จุดนั้นๆ ถ้าทิศทางที่คำนวณได้ไม่เป็นศูนย์ แสดงว่าเราสามารถที่จะเคลื่อนทิศทางต่อไปได้อีก ทำให้ฟังก์ชั่น $E^{(j)}$ ลดลง และทำกรรมวิธีคำนวณซ้ำต่อไปจนกระทั่งได้จุด Optimum

สำหรับค่า Step Size ซึ่งเป็นอัตราการเปลี่ยนแปลงสัมพัทธ์ในสัมประสิทธิ์ $P_i^{(j)}$ หาได้จากสมการ(4.10)ซึ่งมีผลโดยตรงทำให้ค่าผิดพลาดของฟังก์ชั่น $E^{(j)}$ ลดลง เนื่องจากเป็นค่า Positive Scalar และมีทิศทางเดียวกับทิศทางที่คำนวณได้ เพราะฉะนั้นจะต้องมีการควบคุมค่า Step Size อย่างระมัดระวัง เพื่อจะทำให้กรรมวิธี Optimization มีประสิทธิภาพมากที่สุด ถ้าเลือกค่า Step Size ที่ไม่เหมาะสมเช่น มีค่าน้อยไป ทำให้ต้องใช้จำนวน Iterative มากในการคำนวณหาจุด Optimum และใช้เวลานาน ในทำนองเดียวกันถ้าเลือก Step Size มีค่ามากเกินไปผลจะทำให้เกิด Oscillation ในการคำนวณหาค่าผิดพลาดของ

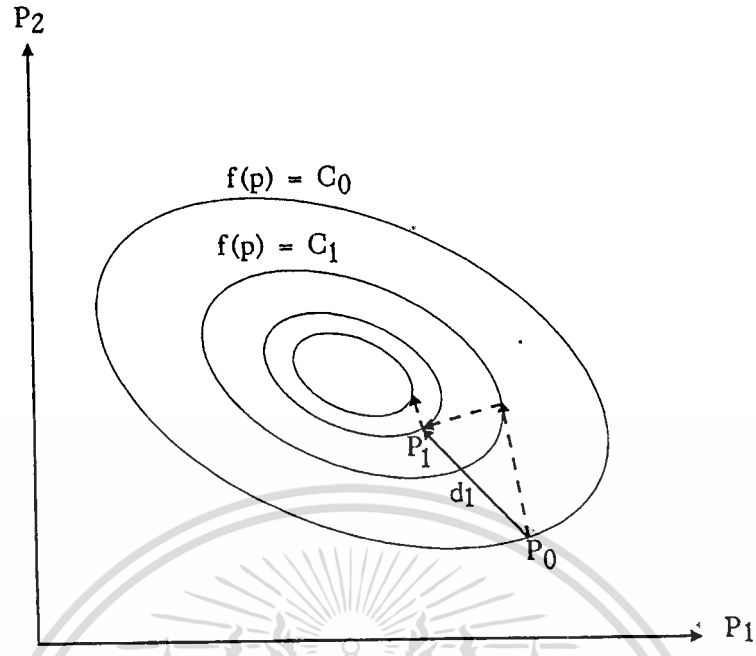
ฟังก์ชั่น $E^{(j)}$ ทำให้ได้จุด Optimum ที่ไม่ดีพอ หรืออาจจะทำให้เกิดความล้มเหลวในกรรมวิธี Optimization ดังรูป 4.7




รูปที่ 4.7 แสดงการเกิด Oscillation โดยใช้วิธี สตีปเปส-ดีเซน

4.2.2 Conjugate Gradient Method

วิธีนี้เป็นผลมาจากการพัฒนาจากวิธี สตีปเปส-ดีเซน ซึ่งทิศทาง สตีปเปส-ดีเซน ในแต่ละการคำนวณซ้ำ ทิศทางจะทำมุมกันและกัน สาเหตุนี้ทำให้วิธี สตีปเปส-ดีเซน นี้ช้าลงในการ Convergent แต่ทิศทางของวิธี คอนจุเกจ-เกรเดียนท์ จะไม่ทำมุมเหมือนกับวิธี สตีปเปส-ดีเซน ในการคำนวณซ้ำ แต่ดูเหมือนว่าทิศทาง คอนจุเกจ-เกรเดียนท์ จะตัดทะแยงมุมผ่านทิศทาง สตีปเปส-ดีเซน เพราะฉะนั้น จะทำให้ Rate Convergence ของวิธี คอนจุเกจ-เกรเดียนท์ มีประสิทธิภาพในกรรมวิธี Optimization ดีขึ้น ดังรูปที่ 4.8




รูปที่ 4.8 แสดงทิศทาง Iterative ของวิธี คอนจุเกจ-เกรเดียนท์

ทิศทางของวิธี คอนจุเกจ-เกรเดียนท์ $d^{(j)}$ ที่จุด $P^{(j)}$ หาได้ดัง

$$d^{(j)} = -\left\{ \frac{\partial E^{(j)}}{\partial P_i^{(j)}} + \bar{B}_k \frac{\partial E^{(j-1)}}{\partial P_i^{(j-1)}} \right\} \qquad \ldots(4.12)$$

โดยที่ $B_k$ เป็น สเกลเฟคเตอร์ (Scale Factor) หาได้ดัง

$$\bar{B}_k = \left\{ \left| \frac{\partial E^{(j)}}{\partial P_i^{(j)}} \right| / \left| \frac{\partial E^{(j-1)}}{\partial P_i^{(j-1)}} \right| \right\}^2 \qquad \ldots(4.13)$$

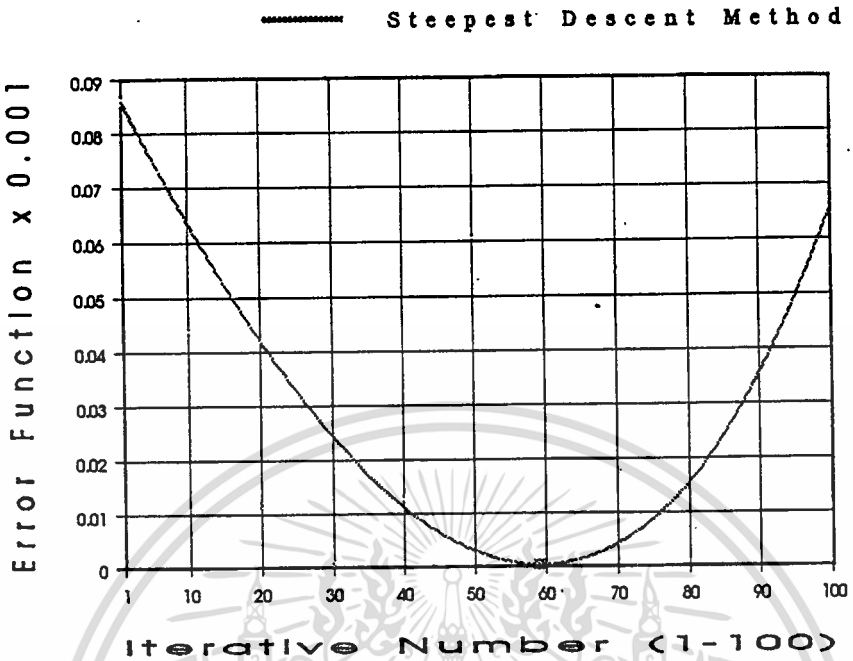
ในการคำนวณซ้ำแต่ละครั้ง การคำนวณครั้งแรกวิธี คอนจุเกจ-เกรเดียนท์ ได้ค่าเช่นเดียวกับการคำนวณครั้งแรกของวิธี สตีปเปส-ดีเซน ข้อแตกต่างกันของทั้งสองวิธีคือสมการที่ (4.12) และ (4.13) ทิศทางของวิธีนี้จะถูกพัฒนาใหม่ โดยใช้ทิศทางที่คำนวณได้ที่ Current Design ($d^{(j)}$) บวกกับทิศทางที่คำนวณได้ครั้งก่อน ($d^{(j-1)}$) ซึ่งจะได้ทิศทางที่ตัดทะแยงมุมของทิศทางทั้งสอง ดังนั้นทิศทาง คอนจุเกจ-เกรเดียนท์ ไม่เพียงทำให้ทิศทาง สตีปเปส-ดีเซน ดีขึ้นเท่านั้น แต่จะเป็นผลทำให้ Rate Covergence ดีขึ้นอย่างมาก

## 4.3 ผลจากการประมาณฟังก์ชั่นด้วยวิธี เน็คกาทีฟ-เกรเดียนท์

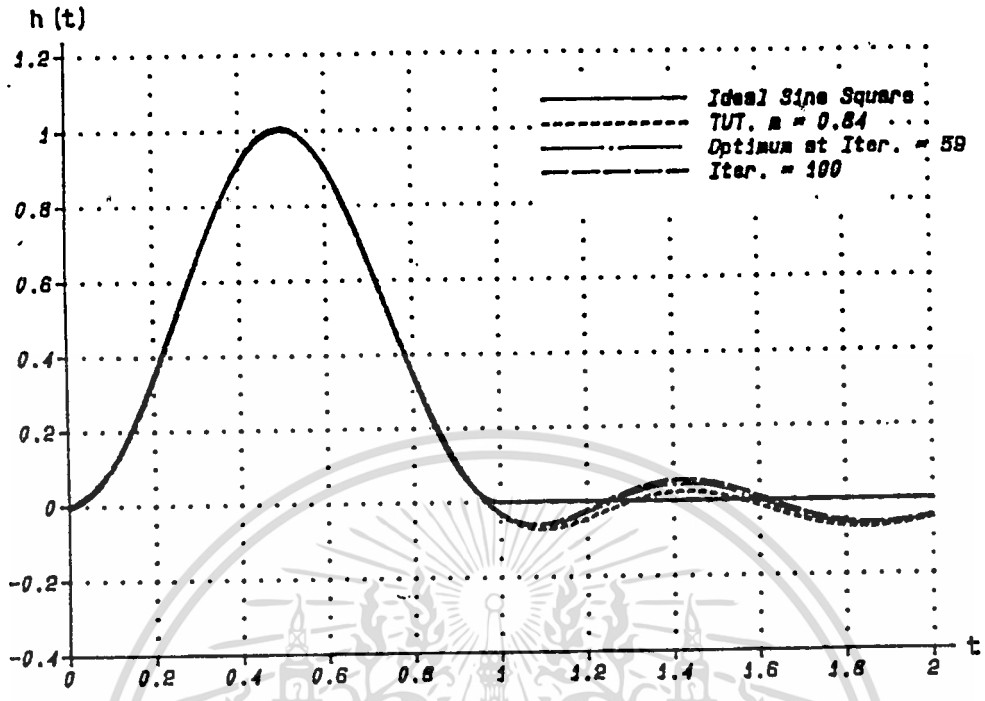
จากกรรมวิธี Optimization ที่ใช้ในการประมาณฟังก์ชั่น ดังโฟว์ชาร์ตที่ 4.4 ซึ่งใช้ทิศทางของวิธี สตีปเปส-ดีเซน และทิศทางของวิธี คอนจุเกจ-เกรเดียนท์ ในรูปที่ 4.6 และรูปที่ 4.8 โดยกำหนดค่าเริ่มต้นด้วย ทรานส์ซิทชั่นนัลอุลตร้าสเฟียริคัล-ทอมป์สันโพลิโนเมียล ที่ m=0.84 ในตารางที่ 3.8 และกำหนดค่า Step Size ที่ k=0.1% จากการคำนวณซ้ำตามวิธีสตีปเปส-ดีเซน ผลที่ได้แสดงไว้ในตารางที่ 4.1 และกราฟรูปที่ 4.1

| Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. | Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. |
|---|---|---|---|---|---|---|---|
| 1 | -2.2327 ±j0.7967 | -1.5701 ± j2.2760 | 0.000,086,024 | 51 | -2.3643 ± j0.7738 | -1.5701 ± j2.2757 | 0.000,002,418 |
| 2 | -2.2351 ±j0.7962 | -1.5701 ± j2.2759 | 0.000,083,416 | 52 | -2.3671 ± j0.7723 | -1.5701 ± j2.2756 | 0.000,001,947 |
| 3 | -2.2376 ±j0.7948 | -1.5701 ± j2.2760 | 0.000,080,838 | 53 | -2.3700 ± j0.7719 | -1.5701 ± j2.2757 | 0.000,001,534 |
| 4 | -2.2400 ±j0.7943 | -1.5701 ± j2.2759 | 0.000,078,292 | 54 | -2.3728 ± j0.7714 | -1.5701 ± j2.2756 | 0.000,001,180 |
| 5 | -2.2425 ±j0.7939 | -1.5701 ± j2.2760 | 0.000,075,778 | 55 | -2.3758 ± j0.7709 | -1.5701 ± j2.2756 | 0.000,000,886 |
| 6 | -2.2449 ±j0.7934 | -1.5701 ± j2.2759 | 0.000,073,297 | 56 | -2.3786 ± j0.7704 | -1.5701 ± j2.2756 | 0.000,000,652 |
| 7 | -2.2475 ±j0.7930 | -1.5701 ± j2.2760 | 0.000,070,874 | 57 | -2.3816 ± j0.7700 | -1.5701 ± j2.2756 | 0.000,000,480 |
| 8 | -2.2499 ±j0.7925 | -1.5701 ± j2.2759 | 0.000,068,431 | 58 | -2.3844 ± j0.7695 | -1.5701 ± j2.2756 | 0.000,000,370 |
| 9 | -2.2524 ±j0.7921 | -1.5701 ± j2.2759 | 0.000,066,048 | 59 | -2.3874 ± j0.7690 | -1.5701 ± j2.2756 | 0.000,000,323 |
| 10 | -2.2549 ±j0.7916 | -1.5701 ± j2.2759 | 0.000,063,699 | 60 | -2.3903 ± j0.7685 | -1.5701 ± j2.2756 | 0.000,000,340 |
| 11 | -2.2574 ±j0.7912 | -1.5701 ± j2.2759 | 0.000,061,385 | 61 | -2.3933 ± j0.7681 | -1.5701 ± j2.2756 | 0.000,000,422 |
| 12 | -2.2599 ±j0.7907 | -1.5701 ± j2.2759 | 0.000,059,105 | 62 | -2.2962 ± j0.7676 | -1.5701 ± j2.2756 | 0.000,000,570 |
| 13 | -2.2625 ±j0.7903 | -1.5701 ± j2.2759 | 0.000,056,860 | 63 | -2.3992 ± j0.7671 | -1.5701 ± j2.2756 | 0.000,000,784 |
| 14 | -2.2649 ±j0.7898 | -1.5701 ± j2.2759 | 0.000,054,650 | 64 | -2.4022 ± j0.7666 | -1.5701 ± j2.2755 | 0.000,001,066 |
| 15 | -2.2675 ±j0.7894 | -1.5701 ± j2.2759 | 0.000,052,477 | 65 | -2.4052 ± j0.7662 | -1.5701 ± j2.2756 | 0.000,001,417 |
| 16 | -2.2700 ±j0.7889 | -1.5701 ± j2.2758 | 0.000,050,340 | 66 | -2.4082 ± j0.7657 | -1.5701 ± j2.2755 | 0.000,001,837 |
| 17 | -2.2726 ±j0.7885 | -1.5701 ± j2.2759 | 0.000,048,239 | 67 | -2.4113 ± j0.7652 | -1.5701 ± j2.2756 | 0.000,002,327 |
| 18 | -2.2751 ±j0.7880 | -1.5701 ± j2.2758 | 0.000,046,176 | 68 | -2.4142 ± j0.7647 | -1.5701 ± j2.2755 | 0.000,002,889 |
| 19 | -2.2777 ±j0.7876 | -1.5701 ± j2.2759 | 0.000,044,151 | 69 | -2.4173 ± j0.7643 | -1.5701 ± j2.2756 | 0.000,003,623 |
| 20 | -2.2803 ±j0.7871 | -1.5701 ± j2.2758 | 0.000,042,164 | 70 | -2.4204 ± j0.7638 | -1.5701 ± j2.2755 | 0.000,004,231 |
| 21 | -2.2829 ±j0.7867 | -1.5701 ± j2.2759 | 0.000,040,216 | 71 | -2.4235 ± j0.7633 | -1.5701 ± j2.2755 | 0.000,005,013 |
| 22 | -2.2855 ±j0.7862 | -1.5701 ± j2.2758 | 0.000,038,307 | 72 | -2.4265 ± j0.7628 | -1.5700 ± j2.2755 | 0.000,005,871 |
| 23 | -2.2881 ±j0.7858 | -1.5701 ± j2.2758 | 0.000,036,437 | 73 | -2.4296 ± j0.7624 | -1.5701 ± j2.2755 | 0.000,006,806 |
| 24 | -2.2906 ±j0.7853 | -1.5701 ± j2.2758 | 0.000,034,608 | 74 | -2.4327 ± j0.7614 | -1.5701 ± j2.2755 | 0.000,007,818 |
| 25 | -2.2933 ±j0.7849 | -1.5701 ± j2.2758 | 0.000,032,819 | 75 | -2.4359 ± j0.7614 | -1.5701 ± j2.2755 | 0.000,008,909 |
| 26 | -2.2959 ±j0.7844 | -1.5701 ± j2.2758 | 0.000,031,072 | 76 | -2.4390 ± j0.7609 | -1.5700 ± j2.2755 | 0.000,010,080 |
| 27 | -2.2985 ±j0.7839 | -1.5701 ± j2.2758 | 0.000,029,366 | 77 | -2.4422 ± j0.7605 | -1.5701 ± j2.2755 | 0.000,011,332 |
| 28 | -2.3011 ±j0.7834 | -1.5701 ± j2.2758 | 0.000,027,702 | 78 | -2.4453 ± j0.7599 | -1.5700 ± j2.2755 | 0.000,012,667 |
| 29 | -2.3038 ±j0.7830 | -1.5701 ± j2.2758 | 0.000,026,081 | 79 | -2.4485 ± j0.7595 | -1.5701 ± j2.2755 | 0.000,014,085 |
| 30 | -2.3064 ±j0.7825 | -1.5701 ± j2.2758 | 0.000,024,504 | 80 | -2.4516 ± j0.7590 | -1.5700 ± j2.2755 | 0.000,015,588 |
| 31 | -2.3091 ±j0.7821 | -1.5701 ± j2.2758 | 0.000,022,970 | 81 | -2.4549 ± j0.7580 | -1.5701 ± j2.2755 | 0.000,017,177 |
| 32 | -2.3118 ±j0.7816 | -1.5701 ± j2.2757 | 0.000,021,481 | 82 | -2.4580 ± j0.7576 | -1.5700 ± j2.2754 | 0.000,018,853 |
| 33 | -2.3145 ±j0.7812 | -1.5701 ± j2.2758 | 0.000,020,036 | 83 | -2.4613 ± j0.7576 | -1.5701 ± j2.2755 | 0.000,020,618 |
| 34 | -2.3171 ±j0.7807 | -1.5701 ± j2.2757 | 0.000,018,637 | 84 | -2.4645 ± j0.7570 | -1.5700 ± j2.2754 | 0.000,022,472 |
| 35 | -2.3199 ±j0.7803 | -1.5701 ± j2.2758 | 0.000,017,284 | 85 | -2.4678 ± j0.7566 | -1.5701 ± j2.2755 | 0.000,024,418 |
| 36 | -2.3225 ±j0.7798 | -1.5701 ± j2.2757 | 0.000,015,978 | 86 | -2.4710 ± j0.7561 | -1.5700 ± j2.2754 | 0.000,026,457 |
| 37 | -2.3253 ±j0.7793 | -1.5701 ± j2.2758 | 0.000,014,719 | 87 | -2.4744 ± j0.7556 | -1.5701 ± j2.2755 | 0.000,028,590 |
| 38 | -2.3280 ±j0.7788 | -1.5701 ± j2.2757 | 0.000,013,508 | 88 | -2.4776 ± j0.7551 | -1.5700 ± j2.2754 | 0.000,030,818 |
| 39 | -2.3307 ±j0.7779 | -1.5701 ± j2.2757 | 0.000,012,346 | 89 | -2.4810 ± j0.7547 | -1.5701 ± j2.2755 | 0.000,033,144 |
| 40 | -2.3334 ±j0.7775 | -1.5701 ± j2.2757 | 0.000,011,233 | 90 | -2.4843 ± j0.7541 | -1.5700 ± j2.2754 | 0.000,035,667 |
| 41 | -2.3362 ±j0.7776 | -1.5701 ± j2.2757 | 0.000,010,169 | 91 | -2.4877 ± j0.7537 | -1.5701 ± j2.2754 | 0.000,038,091 |
| 42 | -2.3390 ±j0.7770 | -1.5701 ± j2.2757 | 0.000,009,156 | 92 | -2.4910 ± j0.7532 | -1.5700 ± j2.2754 | 0.000,040,716 |
| 43 | -2.3418 ±j0.7766 | -1.5701 ± j2.2757 | 0.000,008,194 | 93 | -2.4944 ± j0.7527 | -1.5701 ± j2.2754 | 0.000,043,444 |
| 44 | -2.3445 ±j0.7760 | -1.5701 ± j2.2757 | 0.000,007,284 | 94 | -2.4977 ± j0.7522 | -1.5700 ± j2.2754 | 0.000,046,277 |
| 45 | -2.3474 ±j0.7750 | -1.5701 ± j2.2757 | 0.000,006,426 | 95 | -2.5012 ± j0.7518 | -1.5701 ± j2.2754 | 0.000,049,217 |
| 46 | -2.3501 ±j0.7751 | -1.5701 ± j2.2756 | 0.000,005,621 | 96 | -2.5046 ± j0.7512 | -1.5701 ± j2.2754 | 0.000,052,264 |
| 47 | -2.3529 ±j0.7749 | -1.5701 ± j2.2757 | 0.000,004,870 | 97 | -2.5081 ± j0.7508 | -1.5701 ± j2.2754 | 0.000,055,421 |
| 48 | -2.3557 ±j0.7742 | -1.5701 ± j2.2756 | 0.000,004,173 | 98 | -2.5115 ± j0.7503 | -1.5700 ± j2.2754 | 0.000,058,690 |
| 49 | -2.3586 ±j0.7738 | -1.5701 ± j2.2757 | 0.000,003,532 | 99 | -2.5150 ± j0.7493 | -1.5701 ± j2.2754 | 0.000,062,072 |
| 50 | -2.3614 ±j0.7732 | -1.5701 ± j2.2756 | 0.000,002,946 | 100 | -2.5184 ± j0.7493 | -1.5700 ± j2.2754 | 0.000,066,566 |

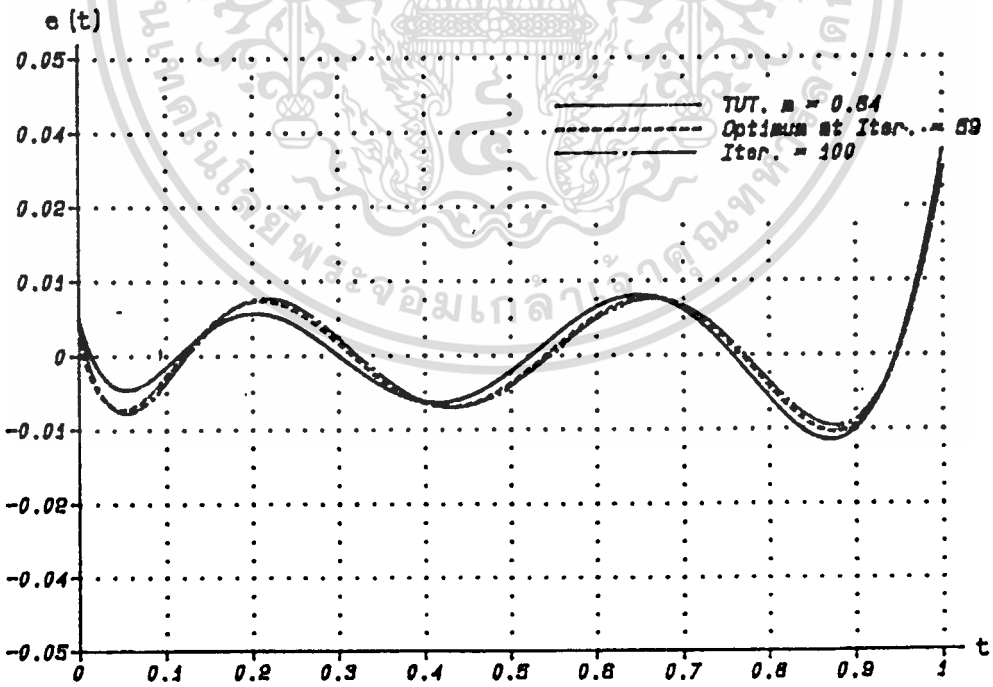
ตารางที่ 4.1 การคำนวณซ้ำโดยใช้วิธี สตีปเปส-ดีเซน ที่ m=0.84,k=0.1%




กราฟที่ 4.2 แสดงการคำนวณซ้ำโดยใช้วิธี สตีปเปส-ดีเซน ที่ m=0.84,k=0.1%




รูปที่ 4.8 ผลตอบสนองจากการประมาณฟังก์ชั่นโดยวิธี สตีปเปส-ดีเซน ที่ m=0.84,k=0.1%

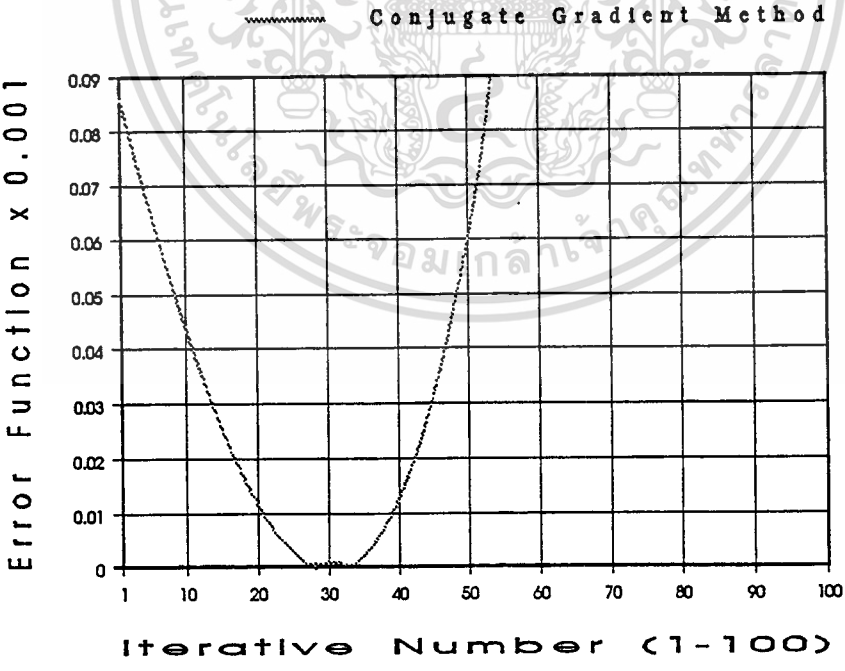



รูปที่ 4.9 ค่าผิดพลาดจากการประมาณฟังก์ชั่นโดยวิธี สตีปเปส-ดีเซน ที่ m=0.84,k=0.1%

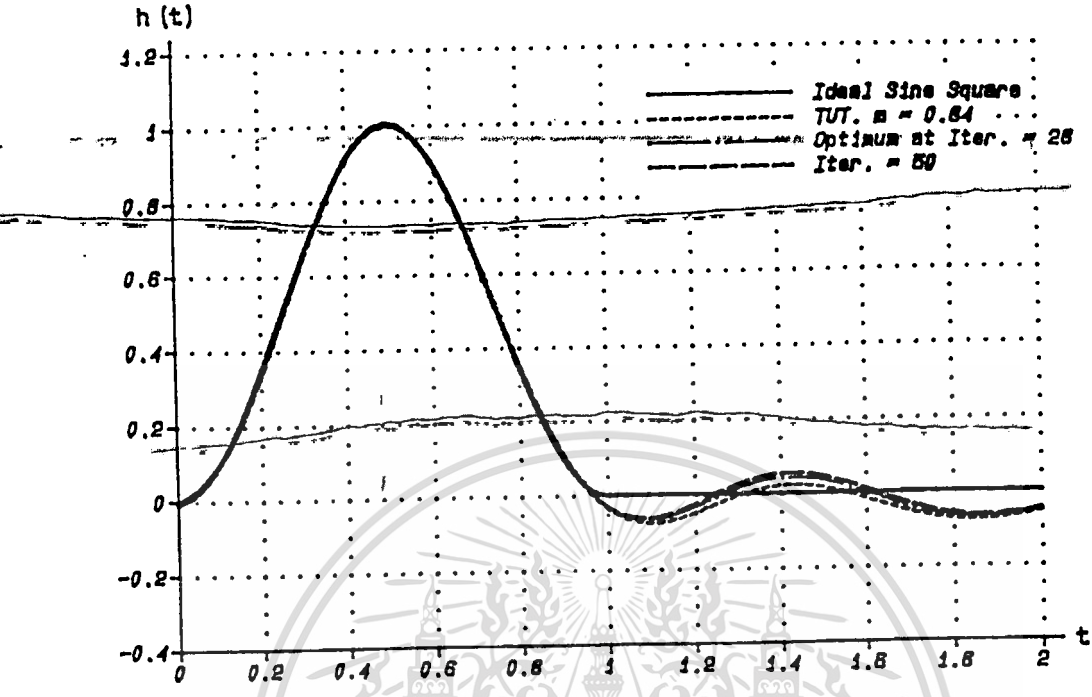
จากการคำนวณซ้ำตามวิธี คอนจุเกจ-เกรเดียนท์ ผลที่ได้ดังแสดงไว้ในตารางที่ 4.2 และกราฟที่ 4.2

| Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. | Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. |
|---|---|---|---|---|---|---|---|
| 1 | -2.2327 ± j0.7957 | -1.5701± j2.2760 | 0.000,086,024 | 26 | -2.3649 ± j0.7726 | -1.5701± j2.2757 | 0.000,001,612 |
| 2 | -2.2376 ± j0.7949 | -1.5701± j2.2759 | 0.000,080,961 | 27 | -2.3707 ± j0.7717 | -1.5701± j2.2757 | 0.000,000,537 |
| 3 | -2.2426 ± j0.7939 | -1.5701± j2.2760 | 0.000,076,851 | 28 | -2.3766 ± j0.7707 | -1.5701± j2.2756 | 0.000,000,148 |
| 4 | -2.2476 ± j0.7929 | -1.5701± j2.2759 | 0.000,070,871 | 29 | -2.3824 ± j0.7698 | -1.5701± j2.2756 | 0.000,000,589 |
| 5 | -2.2526 ± j0.7921 | -1.5701± j2.2759 | 0.000,066,023 | 30 | -2.3882 ± j0.7688 | -1.5701± j2.2756 | 0.000,000,781 |
| 6 | -2.2576 ± j0.7911 | -1.5701± j2.2759 | 0.000,061,311 | 31 | -2.3942 ± j0.7679 | -1.5701± j2.2756 | 0.000,000,715 |
| 7 | -2.2627 ± j0.7903 | -1.5701± j2.2759 | 0.000,056,737 | 32 | -2.4001 ± j0.7669 | -1.5701± j2.2756 | 0.000,000,386 |
| 8 | -2.2677 ± j0.7893 | -1.5701± j2.2758 | 0.000,052,307 | 33 | -2.4061 ± j0.7660 | -1.5701± j2.2756 | 0.000,000,215 |
| 9 | -2.2729 ± j0.7885 | -1.5701± j2.2759 | 0.000,048,021 | 34 | -2.4122 ± j0.7650 | -1.5701± j2.2756 | 0.000,001,096 |
| 10 | -2.2780 ± j0.7875 | -1.5701± j2.2758 | 0.000,043,886 | 35 | -2.4183 ± j0.7641 | -1.5701± j2.2756 | 0.000,002,262 |
| 11 | -2.2832 ± j0.7866 | -1.5701± j2.2759 | 0.000,039,903 | 36 | -2.4244 ± j0.7631 | -1.5701± j2.2755 | 0.000,003,724 |
| 12 | -2.2884 ± j0.7857 | -1.5701± j2.2758 | 0.000,036,078 | 37 | -2.4307 ± j0.7621 | -1.5701± j2.2755 | 0.000,005,490 |
| 13 | -2.2937 ± j0.7848 | -1.5701± j2.2758 | 0.000,032,414 | 38 | -2.4369 ± j0.7611 | -1.5701± j2.2755 | 0.000,007,568 |
| 14 | -2.2989 ± j0.7838 | -1.5701± j2.2758 | 0.000,028,915 | 39 | -2.4433 ± j0.7602 | -1.5701± j2.2755 | 0.000,009,968 |
| 15 | -2.3042 ± j0.7829 | -1.5701± j2.2758 | 0.000,025,585 | 40 | -2.4496 ± j0.7592 | -1.5701± j2.2755 | 0.000,012,698 |
| 16 | -2.3095 ± j0.7820 | -1.5701± j2.2758 | 0.000,022,428 | 41 | -2.4560 ± j0.7583 | -1.5701± j2.2755 | 0.000,015,770 |
| 17 | -2.3150 ± j0.7811 | -1.5701± j2.2758 | 0.000,019,450 | 42 | -2.4625 ± j0.7572 | -1.5701± j2.2755 | 0.000,019,192 |
| 18 | -2.3203 ± j0.7801 | -1.5701± j2.2758 | 0.000,016,654 | 43 | -2.4691 ± j0.7563 | -1.5701± j2.2755 | 0.000,022,976 |
| 19 | -2.3258 ± j0.7792 | -1.5701± j2.2757 | 0.000,014,046 | 44 | -2.4756 ± j0.7553 | -1.5701± j2.2755 | 0.000,027,133 |
| 20 | -2.3312 ± j0.7783 | -1.5701± j2.2757 | 0.000,011,630 | 45 | -2.4823 ± j0.7544 | -1.5701± j2.2754 | 0.000,031,673 |
| 21 | -2.3368 ± j0.7774 | -1.5701± j2.2757 | 0.000,009,411 | 46 | -2.4890 ± j0.7533 | -1.5701± j2.2754 | 0.000,036,609 |
| 22 | -2.3432 ± j0.7764 | -1.5701± j2.2757 | 0.000,007,394 | 47 | -2.4958 ± j0.7524 | -1.5701± j2.2754 | 0.000,041,954 |
| 23 | -2.3480 ± j0.7755 | -1.5701± j2.2757 | 0.000,005,585 | 48 | -2.5026 ± j0.7514 | -1.5701± j2.2754 | 0.000,047,719 |
| 24 | -2.3535 ± j0.7745 | -1.5701± j2.2757 | 0.000,003,989 | 49 | -2.5095 ± j0.7505 | -1.5701± j2.2754 | 0.000,053,919 |
| 25 | -2.3593 ± j0.7736 | -1.5701± j2.2757 | 0.000,002,612 | 50 | -2.5164 ± j0.7494 | -1.5701± j2.2754 | 0.000,060,567 |

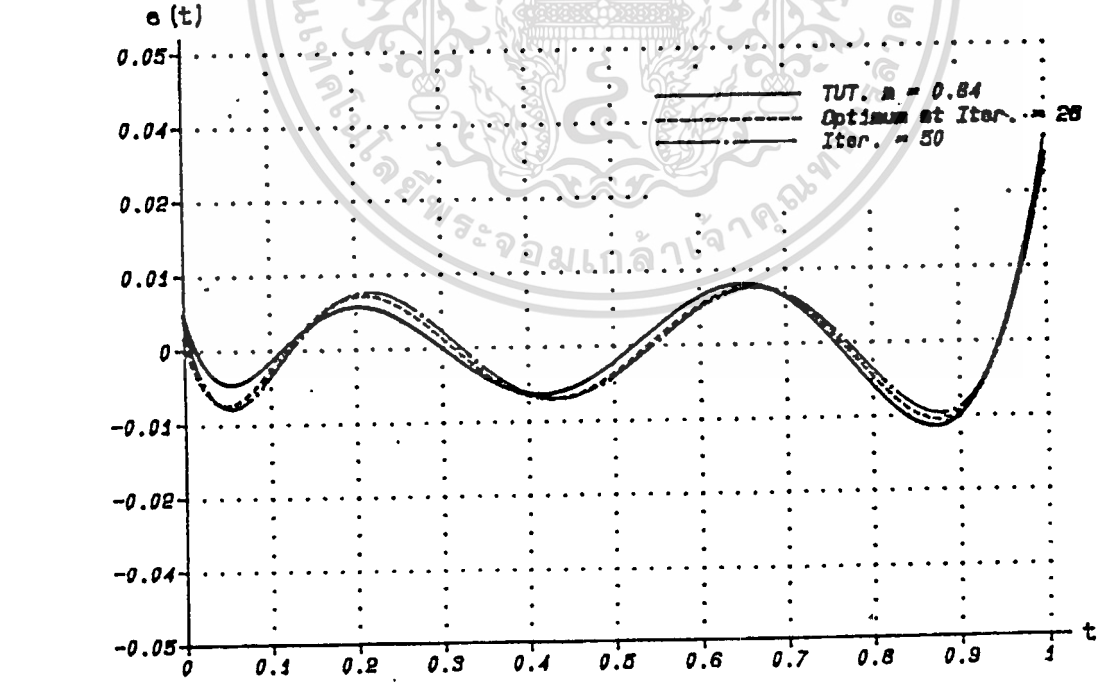
ตารางที่ 4.2 การคำนวณซ้ำโดยใช้วิธี คอนจุเกจ-เกรเดียนท์ ที่ m=0.84,k=0.1%




กราฟที่ 4.2 แสดงการคำนวณซ้ำโดยใช้วิธี คอนจุเกจ-เกรเดียนท์ ที่ m=0.84,k=0.1%




รูปที่ 4.10 ผลตอบสนองจากการประมาณฟังก์ชั่นโดยวิธี คอนจุเกจ-เกรเดียนท์ที่ m=0.84,k=0.1%



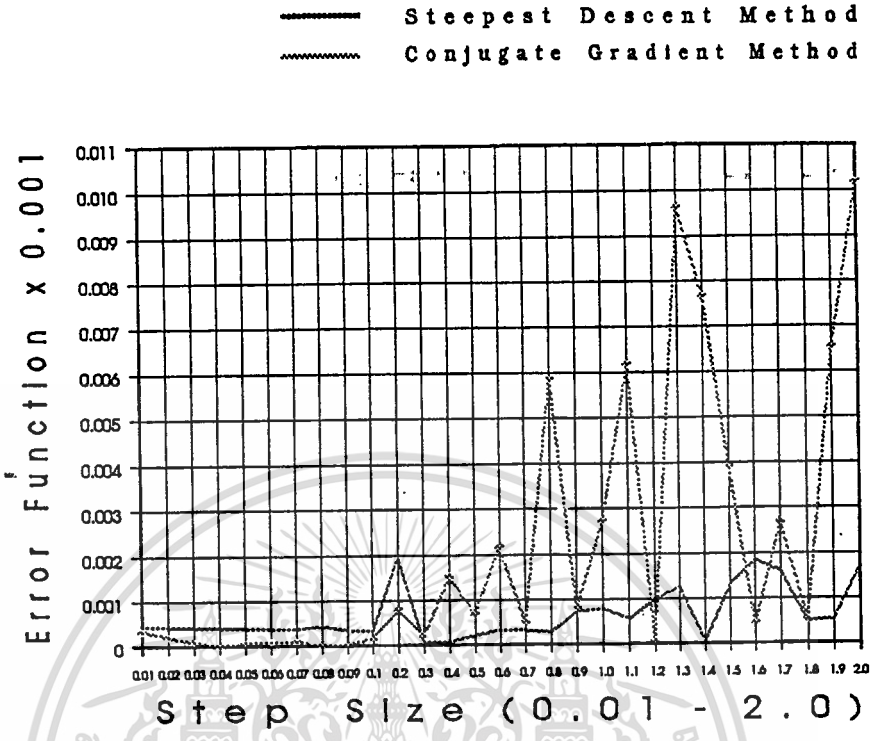

รูปที่ 4.11 ค่าผิดพลาดจากการประมาณฟังก์ชั่นโดยวิธี คอนจุเกจ-เกรเดียนท์ที่ m=0.84,k=0.1%

จากการกำหนดค่า Step Size ที่ k=0.1% และได้ผลจากการคำนวณซ้ำของวิธี สตีปเปส-ดีเซน และวิธี คอนจุเกจ-เกรเดียนท์ดังตารางที่ 4.1 และตารางที่ 4.2 ซึ่งวิธี สตีปเปส-ดีเซนได้จุด Optimum ที่ Iter.=59 ครั้ง Error=0.000,000,323 และวิธี คอนจุเกจ-เกรเดียนท์ ได้จุด Optimum ที่ Iter.=28 ครั้ง Error=0.000,000,148 จะเห็นว่าวิธี คอนจุเกจ-เกรเดียนท์ ให้ Rate Convergence ที่ดีกว่า และให้ค่าผิดพลาดที่น้อยกว่าวิธี สตีปเปส-ดีเซน

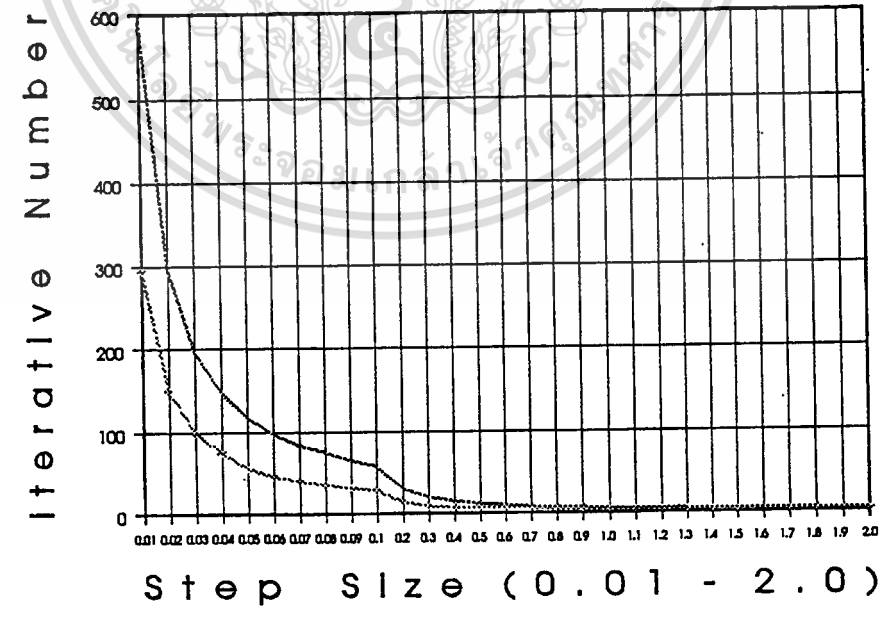
เนื่องจากค่า Step Size ซึ่งเป็นอัตราการเปลี่ยนแปลงสัมพัทธ์ของสัมประสิทธิ์ที่เป็นค่าสเกลล่า เพราะฉะนั้นเราสามารถปรับค่า Step Size ให้มีค่าเพิ่มมากขึ้นหรือลดลงได้ โดยขึ้นอยู่กับการต้องการความละเอียดในการคำนวณซ้ำในกรรมวิธี Optimization และเช่นเดียวกันเรากำหนดค่าเริ่มต้นด้วย ทรานสิชชั่นนัลอุลตร้าสเฟียริคัล-ทอมป์สันโพลิโนเมียล ที่ m=0.84 ในตารางที่ 3.8 และปรับค่า Step Size ที่ k=0.01% ถึง k=2% ซึ่งจากการคำนวณซ้ำตามวิธี สตีปเปส-ดีเซน และวิธี คอนจุเกจ-เกรเดียนท์ เพื่อหาค่า Step Size ที่ทำให้ได้ค่าผิดพลาดน้อยที่สุด ซึ่งเป็นจุด Optimum จากทั้งสองวิธี และผลที่ได้แสดงไว้ในตารางที่ 4.3 และกราฟที่ 4.3

| Step Size (0.01-2) | Steepest Descent Method | | Conjugate Gradient Method | |
|---|---|---|---|---|
| | Iter. | Error. | Iter. | Error. |
| 0.01 | 589 | 0.000,000,436 | 295 | 0.000,000,323 |
| 0.02 | 295 | 0.000,000,423 | 149 | 0.000,000,202 |
| 0.03 | 197 | 0.000,000,411 | 99 | 0.000,000,073 |
| **0.04** | **148** | **0.000,000,400** | **76** | **0.000,000,001** |
| 0.05 | 118 | 0.000,000,385 | 57 | 0.000,000,058 |
| 0.06 | 98 | 0.000,000,375 | 47 | 0.000,000,082 |
| 0.07 | 84 | 0.000,000,363 | 40 | 0.000,000,091 |
| 0.08 | 76 | 0.000,000,433 | 35 | 0.000,000,009 |
| 0.09 | 66 | 0.000,000,335 | 31 | 0.000,000,014 |
| 0.10 | 59 | 0.000,000,323 | 28 | 0.000,000,148 |
| 0.20 | 30 | 0.000,000,198 | 14 | 0.000,000,812 |
| 0.30 | 20 | 0.000,000,060 | 9 | 0.000,000,206 |
| 0.40 | 15 | 0.000,000,074 | 7 | 0.000,001,530 |
| 0.50 | 12 | 0.000,000,204 | 8 | 0.000,000,695 |
| 0.60 | 10 | 0.000,000,330 | 7 | 0.000,002,187 |
| 0.70 | 8 | 0.000,000,316 | 6 | 0.000,000,499 |
| 0.80 | 7 | 0.000,000,269 | 5 | 0.000,005,868 |
| 0.90 | 7 | 0.000,000,736 | 5 | 0.000,000,836 |
| 1.00 | 6 | 0.000,000,797 | 3 | 0.000,002,730 |
| 1.10 | 5 | 0.000,000,554 | 4 | 0.000,006,186 |
| 1.20 | 5 | 0.000,001,009 | 4 | 0.000,000,061 |
| 1.30 | 5 | 0.000,001,268 | 4 | 0.000,009,643 |
| 1.40 | 4 | 0.000,000,048 | 2 | 0.000,007,684 |
| 1.50 | 4 | 0.000,001,302 | 2 | 0.000,003,920 |
| 1.60 | 4 | 0.000,001,861 | 2 | 0.000,000,459 |
| 1.70 | 4 | 0.000,001,600 | 2 | 0.000,002,692 |
| 1.80 | 4 | 0.000,000,489 | 3 | 0.000,000,536 |
| 1.90 | 3 | 0.000,000,546 | 3 | 0.000,006,544 |
| 2.00 | 3 | 0.000,001,725 | 2 | 0.000,010,199 |

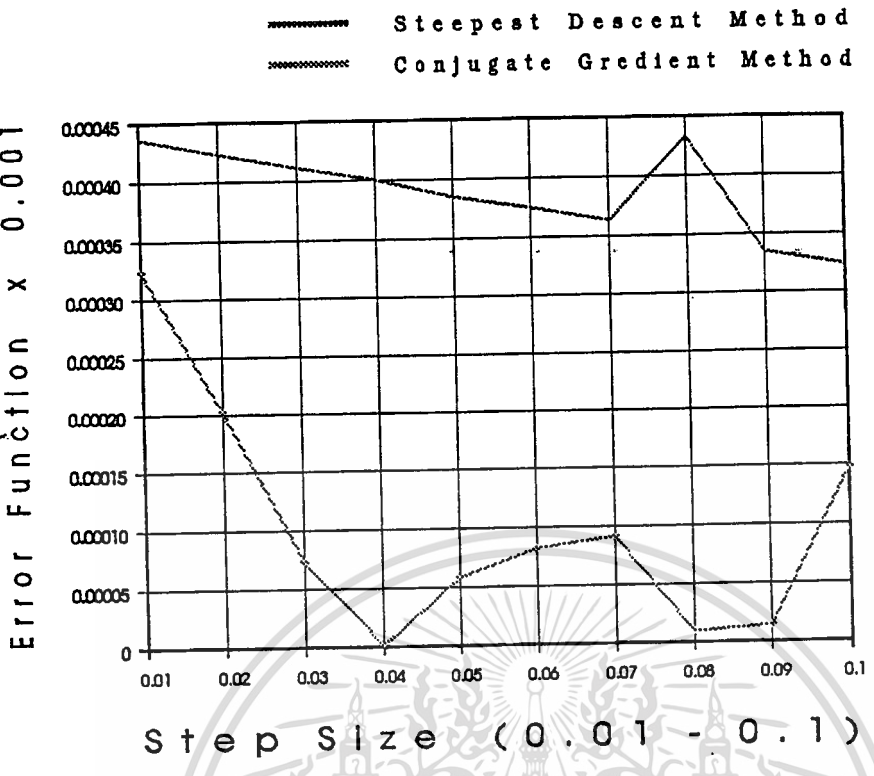
ตารางที่ 4.3 แสดงการคำนวณซ้ำโดยการปรับค่า Step Size ที่ k=0.01% ถึง k=2%



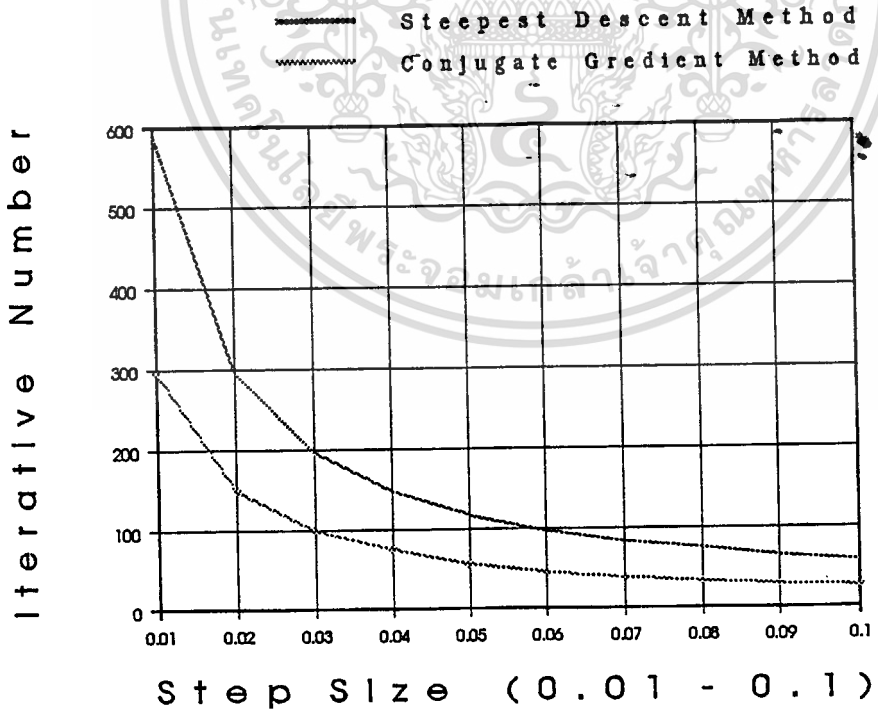

กราฟที่ 4.3 แสดงการคำนวณซ้ำโดยการปรับค่า Step Size ที่ k=0.01% ถึง k=2%




กราฟที่ 4.4 แสดงการคำนวณซ้ำโดยการปรับค่า Step Size ที่ k=0.01% ถึง k=2%




กราฟที่ 4.5 แสดงการคำนวณซ้ำโดยการปรับค่า Step Size ที่ k=0.01% ถึง k=0.1%




กราฟที่ 4.6 แสดงการคำนวณซ้ำโดยการปรับค่า Step Size ที่ k=0.01% ถึง k=0.1%

จากตารางที่ 4.3 และกราฟที่ 4.3 จะเห็นว่าที่ Iter. = 76 Error = 0.000,000,001 ของวิธีคอนจุเกจ-เกรเดียนท์ ให้ค่าผิดพลาดต่ำที่สุดซึ่งเป็นจุด Optimum ที่ Step Size (k)=0.04% ค่ารายละเอียดในการคำนวณซ้ำของวิธี สตีปเปส-ดีเซน และวิธีคอนจุเกจ-เกรเดียนท์ ที่ m=0.84 และ Step Size ที่ k=0.04% ได้ดังตารางที่ 4.4 ตารางที่ 4.5 และตารางที่ 4.6 ส่วนกราฟแสดงการคำนวณซ้ำของทั้งสองวิธีดังกราฟที่ 4.7 และ กราฟที่ 4.8

| Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. | Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. |
|---|---|---|---|---|---|---|---|
| 1 | -2.2312 ±j0.7959 | -1.5701 ± j2.2760 | 0.000,087,356 | 51 | -2.2813 ± j0.7869 | -1.5701 ± j2.2759 | 0.000,041,173 |
| 2 | -2.2322 ±j0.7957 | -1.5701 ± j2.2760 | 0.000,086,301 | 52 | -2.2824 ± j0.7868 | -1.5701 ± j2.2759 | 0.000,040,397 |
| 3 | -2.2331 ±j0.7955 | -1.5701 ± j2.2760 | 0.000,085,252 | 53 | -2.2834 ± j0.7865 | -1.5701 ± j2.2758 | 0.000,039,627 |
| 4 | -2.2341 ±j0.7954 | -1.5701 ± j2.2760 | 0.000,084,208 | 54 | -2.2845 ± j0.7864 | -1.5701 ± j2.2758 | 0.000,038,863 |
| 5 | -2.2351 ±j0.7952 | -1.5701 ± j2.2760 | 0.000,083,169 | 55 | -2.2855 ± j0.7862 | -1.5701 ± j2.2758 | 0.000,038,105 |
| 6 | -2.2361 ±j0.7950 | -1.5701 ± j2.2760 | 0.000,082,135 | 56 | -2.2865 ± j0.7860 | -1.5701 ± j2.2758 | 0.000,037,354 |
| 7 | -2.2370 ±j0.7948 | -1.5701 ± j2.2760 | 0.000,081,105 | 57 | -2.2876 ± j0.7858 | -1.5701 ± j2.2758 | 0.000,036,610 |
| 8 | -2.2380 ±j0.7947 | -1.5701 ± j2.2760 | 0.000,080,081 | 58 | -2.2886 ± j0.7857 | -1.5701 ± j2.2758 | 0.000,035,871 |
| 9 | -2.2390 ±j0.7945 | -1.5701 ± j2.2760 | 0.000,079,062 | 59 | -2.2897 ± j0.7854 | -1.5701 ± j2.2758 | 0.000,035,140 |
| 10 | -2.2400 ±j0.7943 | -1.5701 ± j2.2760 | 0.000,078,048 | 60 | -2.2907 ± j0.7853 | -1.5701 ± j2.2758 | 0.000,034,414 |
| 11 | -2.2410 ±j0.7942 | -1.5701 ± j2.2760 | 0.000,077,039 | 61 | -2.2917 ± j0.7851 | -1.5701 ± j2.2758 | 0.000,033,696 |
| 12 | -2.2420 ±j0.7940 | -1.5701 ± j2.2760 | 0.000,076,035 | 62 | -2.2928 ± j0.7849 | -1.5701 ± j2.2758 | 0.000,032,983 |
| 13 | -2.2429 ±j0.7937 | -1.5701 ± j2.2760 | 0.000,075,036 | 63 | -2.2938 ± j0.7847 | -1.5701 ± j2.2758 | 0.000,032,278 |
| 14 | -2.2440 ±j0.7936 | -1.5701 ± j2.2760 | 0.000,074,043 | 64 | -2.2949 ± j0.7846 | -1.5701 ± j2.2758 | 0.000,031,579 |
| 15 | -2.2449 ±j0.7934 | -1.5701 ± j2.2760 | 0.000,073,055 | 65 | -2.2959 ± j0.7844 | -1.5701 ± j2.2758 | 0.000,030,886 |
| 16 | -2.2459 ±j0.7933 | -1.5701 ± j2.2760 | 0.000,072,072 | 66 | -2.2970 ± j0.7842 | -1.5701 ± j2.2758 | 0.000,030,201 |
| 17 | -2.2469 ±j0.7930 | -1.5701 ± j2.2759 | 0.000,071,094 | 67 | -2.2980 ± j0.7840 | -1.5701 ± j2.2758 | 0.000,029,522 |
| 18 | -2.2479 ±j0.7929 | -1.5701 ± j2.2759 | 0.000,070,121 | 68 | -2.2991 ± j0.7838 | -1.5701 ± j2.2758 | 0.000,028,850 |
| 19 | -2.2489 ±j0.7927 | -1.5701 ± j2.2759 | 0.000,069,154 | 69 | -2.3001 ± j0.7836 | -1.5701 ± j2.2758 | 0.000,028,185 |
| 20 | -2.2499 ±j0.7925 | -1.5701 ± j2.2759 | 0.000,068,192 | 70 | -2.3012 ± j0.7835 | -1.5701 ± j2.2758 | 0.000,027,562 |
| 21 | -2.2509 ±j0.7923 | -1.5701 ± j2.2759 | 0.000,067,236 | 71 | -2.3022 ± j0.7833 | -1.5701 ± j2.2758 | 0.000,026,874 |
| 22 | -2.2519 ±j0.7922 | -1.5701 ± j2.2759 | 0.000,066,285 | 72 | -2.3033 ± j0.7831 | -1.5701 ± j2.2758 | 0.000,026,230 |
| 23 | -2.2529 ±j0.7920 | -1.5701 ± j2.2759 | 0.000,065,339 | 73 | -2.3043 ± j0.7829 | -1.5701 ± j2.2758 | 0.000,025,592 |
| 24 | -2.2539 ±j0.7918 | -1.5701 ± j2.2759 | 0.000,064,399 | 74 | -2.3055 ± j0.7827 | -1.5701 ± j2.2758 | 0.000,024,961 |
| 25 | -2.2549 ±j0.7916 | -1.5701 ± j2.2759 | 0.000,063,464 | 75 | -2.3065 ± j0.7825 | -1.5701 ± j2.2758 | 0.000,024,337 |
| 26 | -2.2599 ±j0.7915 | -1.5701 ± j2.2759 | 0.000,062,535 | 76 | -2.3076 ± j0.7824 | -1.5701 ± j2.2758 | 0.000,023,721 |
| 27 | -2.2569 ±j0.7912 | -1.5701 ± j2.2759 | 0.000,061,612 | 77 | -2.3086 ± j0.7821 | -1.5701 ± j2.2758 | 0.000,023,111 |
| 28 | -2.2579 ±j0.7911 | -1.5701 ± j2.2759 | 0.000,060,693 | 78 | -2.3097 ± j0.7820 | -1.5701 ± j2.2758 | 0.000,022,508 |
| 29 | -2.2589 ±j0.7909 | -1.5701 ± j2.2759 | 0.000,059,781 | 79 | -2.3107 ± j0.7818 | -1.5701 ± j2.2758 | 0.000,021,913 |
| 30 | -2.2599 ±j0.7907 | -1.5701 ± j2.2759 | 0.000,058,874 | 80 | -2.3119 ± j0.7816 | -1.5701 ± j2.2758 | 0.000,021,325 |
| 31 | -2.2609 ±j0.7905 | -1.5701 ± j2.2759 | 0.000,057,973 | 81 | -2.3129 ± j0.7814 | -1.5701 ± j2.2758 | 0.000,020,744 |
| 32 | -2.2620 ±j0.7904 | -1.5701 ± j2.2759 | 0.000,057,077 | 82 | -2.3140 ± j0.7813 | -1.5701 ± j2.2758 | 0.000,020,171 |
| 33 | -2.2629 ±j0.7902 | -1.5701 ± j2.2759 | 0.000,056,187 | 83 | -2.3150 ± j0.7810 | -1.5701 ± j2.2758 | 0.000,019,604 |
| 34 | -2.2640 ±j0.7900 | -1.5701 ± j2.2759 | 0.000,055,303 | 84 | -2.3162 ± j0.7809 | -1.5701 ± j2.2758 | 0.000,019,044 |
| 35 | -2.2649 ±j0.7898 | -1.5701 ± j2.2759 | 0.000,054,424 | 85 | -2.3172 ± j0.7807 | -1.5701 ± j2.2758 | 0.000,018,493 |
| 36 | -2.2660 ±j0.7897 | -1.5701 ± j2.2759 | 0.000,053,552 | 86 | -2.3183 ± j0.7805 | -1.5701 ± j2.2758 | 0.000,017,848 |
| 37 | -2.2670 ±j0.7894 | -1.5701 ± j2.2759 | 0.000,052,685 | 87 | -2.3193 ± j0.7803 | -1.5701 ± j2.2758 | 0.000,017,411 |
| 38 | -2.2680 ±j0.7893 | -1.5701 ± j2.2759 | 0.000,051,824 | 88 | -2.3205 ± j0.7802 | -1.5701 ± j2.2758 | 0.000,016,882 |
| 39 | -2.2690 ±j0.7891 | -1.5701 ± j2.2759 | 0.000,050,968 | 89 | -2.3215 ± j0.7799 | -1.5701 ± j2.2758 | 0.000,016,360 |
| 40 | -2.2701 ±j0.7889 | -1.5701 ± j2.2759 | 0.000,050,119 | 90 | -2.3226 ± j0.7798 | -1.5701 ± j2.2758 | 0.000,015,846 |
| 41 | -2.2710 ±j0.7887 | -1.5701 ± j2.2759 | 0.000,049,276 | 91 | -2.3237 ± j0.7796 | -1.5701 ± j2.2758 | 0.000,015,339 |
| 42 | -2.2721 ±j0.7886 | -1.5701 ± j2.2759 | 0.000,048,438 | 92 | -2.3248 ± j0.7794 | -1.5701 ± j2.2758 | 0.000,014,840 |
| 43 | -2.2731 ±j0.7884 | -1.5701 ± j2.2759 | 0.000,047,607 | 93 | -2.3258 ± j0.7792 | -1.5701 ± j2.2758 | 0.000,014,348 |
| 44 | -2.2742 ±j0.7882 | -1.5701 ± j2.2759 | 0.000,046,781 | 94 | -2.3270 ± j0.7791 | -1.5701 ± j2.2757 | 0.000,013,865 |
| 45 | -2.2751 ±j0.7880 | -1.5701 ± j2.2759 | 0.000,045,962 | 95 | -2.3280 ± j0.7788 | -1.5701 ± j2.2757 | 0.000,013,389 |
| 46 | -2.2762 ±j0.7879 | -1.5701 ± j2.2759 | 0.000,045,148 | 96 | -2.3292 ± j0.7787 | -1.5701 ± j2.2757 | 0.000,012,921 |
| 47 | -2.2772 ±j0.7876 | -1.5701 ± j2.2759 | 0.000,044,341 | 97 | -2.3302 ± j0.7785 | -1.5701 ± j2.2757 | 0.000,012,460 |
| 48 | -2.2783 ±j0.7875 | -1.5701 ± j2.2759 | 0.000,043,640 | 98 | -2.3314 ± j0.7783 | -1.5701 ± j2.2757 | 0.000,012,008 |
| 49 | -2.2792 ±j0.7873 | -1.5701 ± j2.2759 | 0.000,042,745 | 99 | -2.3324 ± j0.7781 | -1.5701 ± j2.2757 | 0.000,011,563 |
| 50 | -2.2803 ±j0.7871 | -1.5701 ± j2.2759 | 0.000,041,956 | 100 | -2.3335 ± j0.7779 | -1.5701 ± j2.2757 | 0.000,011,127 |

ตารางที่ 4.4 แสดงการคำนวณซ้ำด้วยวิธี สตีปเปส-ดีเซน ที่ k=0.04%

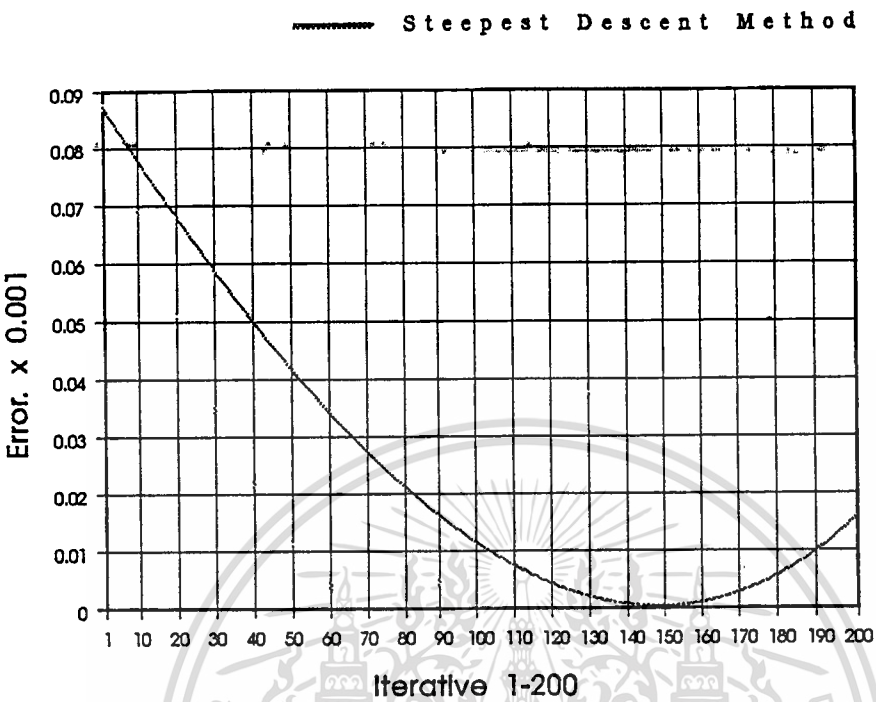
| Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. |
|---|---|---|---|
| 101 | -2.3346 ± j0.7777 | -1.5701 ± j2.2757 | 0.000,010,689 |
| 102 | -2.3358 ± j0.7776 | -1.5701 ± j2.2757 | 0.000,010,278 |
| 103 | -2.3368 ± j0.7773 | -1.5701 ± j2.2757 | 0.000,009,865 |
| 104 | -2.3380 ± j0.7772 | -1.5701 ± j2.2757 | 0.000,009,461 |
| 105 | -2.3391 ± j0.7770 | -1.5701 ± j2.2757 | 0.000,009,066 |
| 106 | -2.3402 ± j0.7768 | -1.5701 ± j2.2757 | 0.000,008,677 |
| 107 | -2.3412 ± j0.7766 | -1.5701 ± j2.2757 | 0.000,008,297 |
| 108 | -2.3424 ± j0.7764 | -1.5701 ± j2.2757 | 0.000,007,926 |
| 109 | -2.3435 ± j0.7762 | -1.5701 ± j2.2757 | 0.000,007,562 |
| 110 | -2.3446 ± j0.7761 | -1.5701 ± j2.2757 | 0.000,007,208 |
| 111 | -2.3457 ± j0.7758 | -1.5701 ± j2.2757 | 0.000,006,861 |
| 112 | -2.3468 ± j0.7757 | -1.5701 ± j2.2757 | 0.000,006,524 |
| 113 | -2.3479 ± j0.7755 | -1.5701 ± j2.2757 | 0.000,006,194 |
| 114 | -2.3491 ± j0.7753 | -1.5701 ± j2.2757 | 0.000,005,874 |
| 115 | -2.3502 ± j0.7751 | -1.5701 ± j2.2757 | 0.000,005,561 |
| 116 | -2.3513 ± j0.7750 | -1.5701 ± j2.2757 | 0.000,005,258 |
| 117 | -2.3524 ± j0.7747 | -1.5701 ± j2.2757 | 0.000,004,963 |
| 118 | -2.3536 ± j0.7746 | -1.5701 ± j2.2757 | 0.000,004,677 |
| 119 | -2.3547 ± j0.7743 | -1.5701 ± j2.2757 | 0.000,004,400 |
| 120 | -2.3558 ± j0.7742 | -1.5701 ± j2.2757 | 0.000,004,131 |
| 121 | -2.3569 ± j0.7740 | -1.5701 ± j2.2757 | 0.000,003,872 |
| 122 | -2.3581 ± j0.7738 | -1.5701 ± j2.2757 | 0.000,003,621 |
| 123 | -2.3592 ± j0.7736 | -1.5701 ± j2.2757 | 0.000,003,380 |
| 124 | -2.3604 ± j0.7735 | -1.5701 ± j2.2757 | 0.000,003,147 |
| 125 | -2.3615 ± j0.7732 | -1.5701 ± j2.2757 | 0.000,002,923 |
| 126 | -2.3626 ± j0.7731 | -1.5701 ± j2.2757 | 0.000,002,709 |
| 127 | -2.3637 ± j0.7730 | -1.5701 ± j2.2757 | 0.000,002,504 |
| 128 | -2.3649 ± j0.7727 | -1.5701 ± j2.2757 | 0.000,002,308 |
| 129 | -2.3660 ± j0.7726 | -1.5701 ± j2.2757 | 0.000,002,121 |
| 130 | -2.3672 ± j0.7723 | -1.5701 ± j2.2757 | 0.000,001,943 |
| 131 | -2.3683 ± j0.7721 | -1.5701 ± j2.2757 | 0.000,001,775 |
| 132 | -2.3695 ± j0.7720 | -1.5701 ± j2.2757 | 0.000,001,616 |
| 133 | -2.3706 ± j0.7717 | -1.5701 ± j2.2757 | 0.000,001,467 |
| 134 | -2.3718 ± j0.7716 | -1.5701 ± j2.2757 | 0.000,001,327 |
| 135 | -2.3730 ± j0.7714 | -1.5701 ± j2.2757 | 0.000,001,197 |
| 136 | -2.3741 ± j0.7712 | -1.5701 ± j2.2757 | 0.000,001,077 |
| 137 | -2.3752 ± j0.7710 | -1.5701 ± j2.2757 | 0.000,000,966 |
| 138 | -2.3764 ± j0.7708 | -1.5701 ± j2.2757 | 0.000,000,864 |
| 139 | -2.3776 ± j0.7706 | -1.5701 ± j2.2756 | 0.000,000,773 |
| 140 | -2.3788 ± j0.7704 | -1.5701 ± j2.2756 | 0.000,000,691 |
| 141 | -2.3799 ± j0.7702 | -1.5701 ± j2.2756 | 0.000,000,620 |
| 142 | -2.3811 ± j0.7701 | -1.5701 ± j2.2756 | 0.000,000,558 |
| 143 | -2.3822 ± j0.7698 | -1.5701 ± j2.2756 | 0.000,000,506 |
| 144 | -2.3834 ± j0.7697 | -1.5701 ± j2.2756 | 0.000,000,464 |
| 145 | -2.3846 ± j0.7695 | -1.5701 ± j2.2756 | 0.000,000,433 |
| 146 | -2.3858 ± j0.7693 | -1.5701 ± j2.2756 | 0.000,000,411 |
| 147 | -2.3869 ± j0.7691 | -1.5701 ± j2.2756 | 0.000,000,400 |
| 148 | -2.3881 ± j0.7689 | -1.5701 ± j2.2756 | 0.000,000,399 |
| 149 | -2.3893 ± j0.7687 | -1.5701 ± j2.2756 | 0.000,000,408 |
| 150 | -2.3905 ± j0.7686 | -1.5701 ± j2.2756 | 0.000,000,428 |

| Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. |
|---|---|---|---|
| 151 | -2.3916 ± j0.7683 | -1.5701 ± j2.2756 | 0.000,000,458 |
| 152 | -2.3928 ± j0.7682 | -1.5701 ± j2.2756 | 0.000,000,498 |
| 153 | -2.3940 ± j0.7679 | -1.5701 ± j2.2756 | 0.000,000,549 |
| 154 | -2.3952 ± j0.7678 | -1.5701 ± j2.2756 | 0.000,000,611 |
| 155 | -2.3964 ± j0.7676 | -1.5701 ± j2.2756 | 0.000,000,683 |
| 156 | -2.3976 ± j0.7674 | -1.5701 ± j2.2756 | 0.000,000,766 |
| 157 | -2.3988 ± j0.7672 | -1.5701 ± j2.2756 | 0.000,000,860 |
| 158 | -2.4000 ± j0.7670 | -1.5701 ± j2.2756 | 0.000,000,965 |
| 159 | -2.4011 ± j0.7668 | -1.5701 ± j2.2756 | 0.000,001,081 |
| 160 | -2.4024 ± j0.7667 | -1.5701 ± j2.2756 | 0.000,001,207 |
| 161 | -2.4036 ± j0.7664 | -1.5701 ± j2.2756 | 0.000,001,345 |
| 162 | -2.4048 ± j0.7663 | -1.5701 ± j2.2756 | 0.000,001,493 |
| 163 | -2.4059 ± j0.7661 | -1.5701 ± j2.2756 | 0.000,001,653 |
| 164 | -2.4072 ± j0.7659 | -1.5701 ± j2.2756 | 0.000,001,824 |
| 165 | -2.4084 ± j0.7657 | -1.5701 ± j2.2756 | 0.000,002,006 |
| 166 | -2.4096 ± j0.7655 | -1.5701 ± j2.2756 | 0.000,002,200 |
| 167 | -2.4108 ± j0.7653 | -1.5701 ± j2.2756 | 0.000,002,405 |
| 168 | -2.4120 ± j0.7651 | -1.5701 ± j2.2756 | 0.000,002,621 |
| 169 | -2.4132 ± j0.7649 | -1.5701 ± j2.2756 | 0.000,002,849 |
| 170 | -2.4145 ± j0.7647 | -1.5701 ± j2.2756 | 0.000,003,089 |
| 171 | -2.4157 ± j0.7645 | -1.5701 ± j2.2756 | 0.000,003,340 |
| 172 | -2.4169 ± j0.7644 | -1.5701 ± j2.2756 | 0.000,003,603 |
| 173 | -2.4181 ± j0.7642 | -1.5701 ± j2.2756 | 0.000,003,878 |
| 174 | -2.4193 ± j0.7640 | -1.5701 ± j2.2756 | 0.000,004,164 |
| 175 | -2.4206 ± j0.7638 | -1.5701 ± j2.2756 | 0.000,004,463 |
| 176 | -2.4218 ± j0.7636 | -1.5701 ± j2.2756 | 0.000,004,773 |
| 177 | -2.4230 ± j0.7634 | -1.5701 ± j2.2756 | 0.000,005,096 |
| 178 | -2.4243 ± j0.7632 | -1.5701 ± j2.2756 | 0.000,005,430 |
| 179 | -2.4255 ± j0.7630 | -1.5701 ± j2.2756 | 0.000,005,777 |
| 180 | -2.4267 ± j0.7628 | -1.5701 ± j2.2756 | 0.000,006,136 |
| 181 | -2.4279 ± j0.7626 | -1.5701 ± j2.2756 | 0.000,006,508 |
| 182 | -2.4292 ± j0.7626 | -1.5701 ± j2.2756 | 0.000,006,892 |
| 183 | -2.4305 ± j0.7626 | -1.5701 ± j2.2756 | 0.000,007,288 |
| 184 | -2.4317 ± j0.7621 | -1.5701 ± j2.2756 | 0.000,007,621 |
| 185 | -2.4329 ± j0.7618 | -1.5701 ± j2.2756 | 0.000,008,118 |
| 186 | -2.4342 ± j0.7617 | -1.5701 ± j2.2756 | 0.000,008,553 |
| 187 | -2.4354 ± j0.7615 | -1.5701 ± j2.2756 | 0.000,008,999 |
| 188 | -2.4367 ± j0.7613 | -1.5701 ± j2.2756 | 0.000,009,459 |
| 189 | -2.4379 ± j0.7611 | -1.5701 ± j2.2756 | 0.000,009,932 |
| 190 | -2.4392 ± j0.7609 | -1.5701 ± j2.2756 | 0.000,010,418 |
| 191 | -2.4404 ± j0.7607 | -1.5701 ± j2.2756 | 0.000,010,916 |
| 192 | -2.4417 ± j0.7605 | -1.5701 ± j2.2756 | 0.000,011,428 |
| 193 | -2.4430 ± j0.7603 | -1.5701 ± j2.2756 | 0.000,011,953 |
| 194 | -2.4438 ± j0.7601 | -1.5701 ± j2.2756 | 0.000,012,736 |
| 195 | -2.4455 ± j0.7600 | -1.5701 ± j2.2756 | 0.000,013,043 |
| 196 | -2.4468 ± j0.7598 | -1.5701 ± j2.2756 | 0.000,013,609 |
| 197 | -2.4481 ± j0.7596 | -1.5701 ± j2.2756 | 0.000,014,187 |
| 198 | -2.4493 ± j0.7593 | -1.5701 ± j2.2756 | 0.000,014,780 |
| 199 | -2.4506 ± j0.7592 | -1.5701 ± j2.2756 | 0.000,015,386 |
| 200 | -2.4519 ± j0.7596 | -1.5701 ± j2.2756 | 0.000,016,006 |

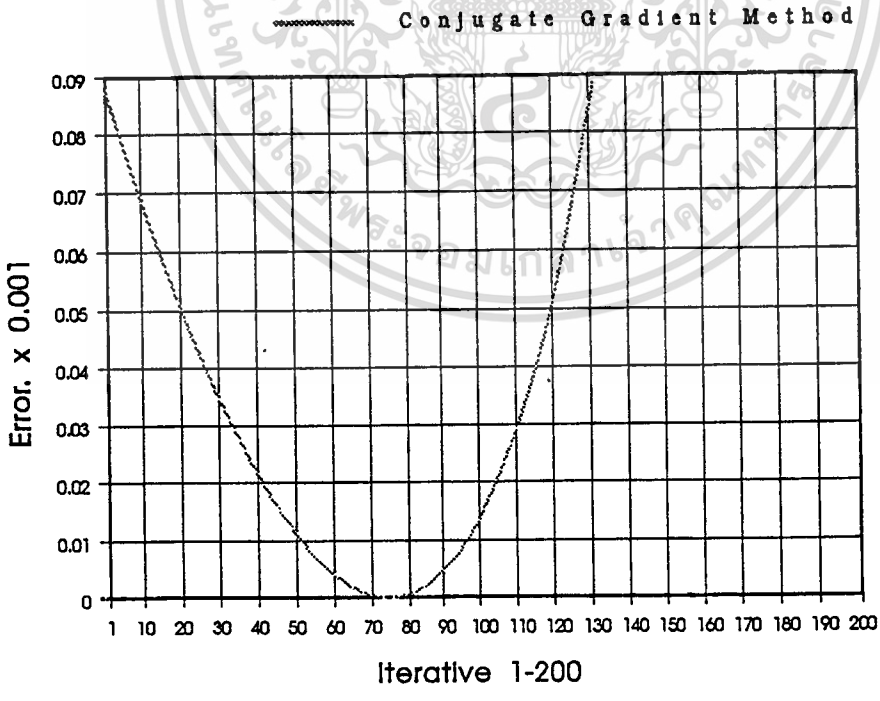
ตารางที่ 4.5 แสดงการคำนวณซ้ำด้วยวิธี สตีปเปส-ดีเซน ที่ k=0.04%

| Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. | Iter. | Pole1-Pole2 | Pole3-Pole4 | Error. |
|---|---|---|---|---|---|---|---|
| 1 | -2.2312 ±j0.7959 | -1.5701 ± j2.2760 | 0.000,087,356 | 51 | -2.3349 ± j0.7777 | -1.5701 ± j2.2757 | 0.000,010,391 |
| 2 | -2.2331 ±j0.7955 | -1.5701 ± j2.2760 | 0.000,085,308 | 52 | -2.3370 ± j0.7773 | -1.5701 ± j2.2757 | 0.000,009,551 |
| 3 | -2.2351 ±j0.7952 | -1.5701 ± j2.2760 | 0.000,083,216 | 53 | -2.3393 ± j0.7770 | -1.5701 ± j2.2757 | 0.000,008,744 |
| 4 | -2.2371 ±j0.7948 | -1.5701 ± j2.2760 | 0.000,081,145 | 54 | -2.3415 ± j0.7765 | -1.5701 ± j2.2757 | 0.000,007,970 |
| 5 | -2.2391 ±j0.7945 | -1.5701 ± j2.2760 | 0.000,079,094 | 55 | -2.3438 ± j0.7762 | -1.5701 ± j2.2757 | 0.000,007,229 |
| 6 | -2.2410 ±j0.7941 | -1.5701 ± j2.2760 | 0.000,077,063 | 56 | -2.3459 ± j0.7758 | -1.5701 ± j2.2757 | 0.000,006,522 |
| 7 | -2.2430 ±j0.7938 | -1.5701 ± j2.2760 | 0.000,075,053 | 57 | -2.3482 ± j0.7755 | -1.5701 ± j2.2757 | 0.000,005,848 |
| 8 | -2.2450 ±j0.7934 | -1.5701 ± j2.2760 | 0.000,073,063 | 58 | -2.3504 ± j0.7750 | -1.5701 ± j2.2756 | 0.000,005,209 |
| 9 | -2.2470 ±j0.7931 | -1.5701 ± j2.2760 | 0.000,071,095 | 59 | -2.3874 ± j0.7690 | -1.5701 ± j2.2756 | 0.000,000,323 |
| 10 | -2.2489 ±j0.7927 | -1.5701 ± j2.2759 | 0.000,069,147 | 60 | -2.3549 ± j0.7743 | -1.5701 ± j2.2756 | 0.000,004,035 |
| 11 | -2.2510 ±j0.7924 | -1.5701 ± j2.2759 | 0.000,067,221 | 61 | -2.3572 ± j0.7740 | -1.5701 ± j2.2756 | 0.000,003,500 |
| 12 | -2.2529 ±j0.7919 | -1.5701 ± j2.2759 | 0.000,065,317 | 62 | -2.3595 ± j0.7735 | -1.5701 ± j2.2756 | 0.000,003,002 |
| 13 | -2.2550 ±j0.7916 | -1.5701 ± j2.2759 | 0.000,063,434 | 63 | -2.3618 ± j0.7732 | -1.5701 ± j2.2756 | 0.000,002,540 |
| 14 | -2.2570 ±j0.7912 | -1.5701 ± j2.2759 | 0.000,061,574 | 64 | -2.3640 ± j0.7728 | -1.5701 ± j2.2756 | 0.000,002,114 |
| 15 | -2.2590 ±j0.7909 | -1.5701 ± j2.2759 | 0.000,059,735 | 65 | -2.3664 ± j0.7725 | -1.5701 ± j2.2756 | 0.000,001,725 |
| 16 | -2.2610 ±j0.7905 | -1.5701 ± j2.2759 | 0.000,057,919 | 66 | -2.3686 ± j0.7720 | -1.5701 ± j2.2756 | 0.000,001,373 |
| 17 | -2.2631 ±j0.7902 | -1.5701 ± j2.2759 | 0.000,056,126 | 67 | -2.3710 ± j0.7717 | -1.5701 ± j2.2756 | 0.000,001,059 |
| 18 | -2.2650 ±j0.7898 | -1.5701 ± j2.2759 | 0.000,054,356 | 68 | -2.3732 ± j0.7713 | -1.5701 ± j2.2756 | 0.000,000,783 |
| 19 | -2.2671 ±j0.7896 | -1.5701 ± j2.2759 | 0.000,052,608 | 69 | -2.3756 ± j0.7709 | -1.5701 ± j2.2756 | 0.000,000,546 |
| 20 | -2.2691 ±j0.7891 | -1.5701 ± j2.2759 | 0.000,050,884 | 70 | -2.3779 ± j0.7705 | -1.5701 ± j2.2756 | 0.000,000,348 |
| 21 | -2.2712 ±j0.7887 | -1.5701 ± j2.2759 | 0.000,049,184 | 71 | -2.3802 ± j0.7702 | -1.5701 ± j2.2756 | 0.000,000,189 |
| 22 | -2.2732 ±j0.7883 | -1.5701 ± j2.2759 | 0.000,047,507 | 72 | -2.3825 ± j0.7698 | -1.5701 ± j2.2756 | 0.000,000,070 |
| 23 | -2.2753 ±j0.7880 | -1.5701 ± j2.2759 | 0.000,045,855 | 73 | -2.3849 ± j0.7694 | -1.5701 ± j2.2756 | 0.000,000,009 |
| 24 | -2.2773 ±j0.7876 | -1.5701 ± j2.2758 | 0.000,044,227 | 74 | -2.3872 ± j0.7690 | -1.5701 ± j2.2756 | 0.000,000,048 |
| 25 | -2.2794 ±j0.7873 | -1.5701 ± j2.2759 | 0.000,042,623 | 75 | -2.3896 ± j0.7687 | -1.5701 ± j2.2756 | 0.000,000,046 |
| 26 | -2.2814 ±j0.7869 | -1.5701 ± j2.2758 | 0.000,041,044 | 76 | -2.3920 ± j0.7682 | -1.5701 ± j2.2756 | 0.000,000,001 |
| 27 | -2.2836 ±j0.7866 | -1.5701 ± j2.2758 | 0.000,039,490 | 77 | -2.3944 ± j0.7679 | -1.5701 ± j2.2756 | 0.000,000,086 |
| 28 | -2.2856 ±j0.7862 | -1.5701 ± j2.2758 | 0.000,037,961 | 78 | -2.3967 ± j0.7675 | -1.5701 ± j2.2756 | 0.000,000,215 |
| 29 | -2.2877 ±j0.7858 | -1.5701 ± j2.2758 | 0.000,036,458 | 79 | -2.3991 ± j0.7671 | -1.5701 ± j2.2756 | 0.000,000,386 |
| 30 | -2.2898 ±j0.7854 | -1.5701 ± j2.2758 | 0.000,034,981 | 80 | -2.4015 ± j0.7667 | -1.5701 ± j2.2756 | 0.000,000,601 |
| 31 | -2.2919 ±j0.7851 | -1.5701 ± j2.2758 | 0.000,033,529 | 81 | -2.4039 ± j0.7664 | -1.5701 ± j2.2756 | 0.000,000,861 |
| 32 | -2.2939 ±j0.7847 | -1.5701 ± j2.2758 | 0.000,032,104 | 82 | -2.4063 ± j0.7660 | -1.5701 ± j2.2756 | 0.000,001,164 |
| 33 | -2.2961 ±j0.7844 | -1.5701 ± j2.2758 | 0.000,030,706 | 83 | -2.4088 ± j0.7656 | -1.5701 ± j2.2756 | 0.000,001,513 |
| 34 | -2.2982 ±j0.7840 | -1.5701 ± j2.2758 | 0.000,029,334 | 84 | -2.4112 ± j0.7652 | -1.5701 ± j2.2756 | 0.000,001,906 |
| 35 | -2.3003 ±j0.7836 | -1.5701 ± j2.2758 | 0.000,027,989 | 85 | -2.4136 ± j0.7649 | -1.5701 ± j2.2756 | 0.000,002,346 |
| 36 | -2.3024 ±j0.7832 | -1.5701 ± j2.2758 | 0.000,026,672 | 86 | -2.4160 ± j0.7644 | -1.5701 ± j2.2756 | 0.000,002,832 |
| 37 | -2.3046 ±j0.7829 | -1.5701 ± j2.2758 | 0.000,025,382 | 87 | -2.4185 ± j0.7641 | -1.5701 ± j2.2756 | 0.000,003,365 |
| 38 | -2.3066 ±j0.7825 | -1.5701 ± j2.2758 | 0.000,024,120 | 88 | -2.4209 ± j0.7637 | -1.5701 ± j2.2755 | 0.000,003,946 |
| 39 | -2.3088 ±j0.7822 | -1.5701 ± j2.2758 | 0.000,022,887 | 89 | -2.4234 ± j0.7633 | -1.5701 ± j2.2755 | 0.000,004,575 |
| 40 | -2.3109 ±j0.7817 | -1.5701 ± j2.2758 | 0.000,021,682 | 90 | -2.4259 ± j0.7629 | -1.5701 ± j2.2755 | 0.000,005,252 |
| 41 | -2.3131 ±j0.7814 | -1.5701 ± j2.2758 | 0.000,020,506 | 91 | -2.4284 ± j0.7625 | -1.5701 ± j2.2755 | 0.000,005,978 |
| 42 | -2.3152 ±j0.7810 | -1.5701 ± j2.2758 | 0.000,019,358 | 92 | -2.4387 ± j0.7621 | -1.5701 ± j2.2755 | 0.000,006,754 |
| 43 | -2.3174 ±j0.7807 | -1.5701 ± j2.2758 | 0.000,018,240 | 93 | -2.4334 ± j0.7618 | -1.5701 ± j2.2755 | 0.000,007,581 |
| 44 | -2.3195 ±j0.7803 | -1.5701 ± j2.2757 | 0.000,017,152 | 94 | -2.4358 ± j0.7614 | -1.5701 ± j2.2755 | 0.000,008,458 |
| 45 | -2.3217 ±j0.7799 | -1.5701 ± j2.2757 | 0.000,016,094 | 95 | -2.4384 ± j0.7610 | -1.5701 ± j2.2755 | 0.000,009,386 |
| 46 | -2.3239 ±j0.7795 | -1.5701 ± j2.2757 | 0.000,015,066 | 96 | -2.4409 ± j0.7606 | -1.5701 ± j2.2755 | 0.000,010,367 |
| 47 | -2.3261 ±j0.7792 | -1.5701 ± j2.2757 | 0.000,014,068 | 97 | -2.4434 ± j0.7602 | -1.5701 ± j2.2755 | 0.000,011,401 |
| 48 | -2.3282 ±j0.7788 | -1.5701 ± j2.2757 | 0.000,013,102 | 98 | -2.4459 ± j0.7598 | -1.5701 ± j2.2755 | 0.000,012,487 |
| 49 | -2.3305 ±j0.7784 | -1.5701 ± j2.2757 | 0.000,012,167 | 99 | -2.4485 ± j0.7595 | -1.5701 ± j2.2755 | 0.000,013,628 |
| 50 | -2.3326 ±j0.7780 | -1.5701 ± j2.2767 | 0.000,011,263 | 100 | -2.4510 ± j0.7590 | -1.5701 ± j2.2755 | 0.000,014,823 |

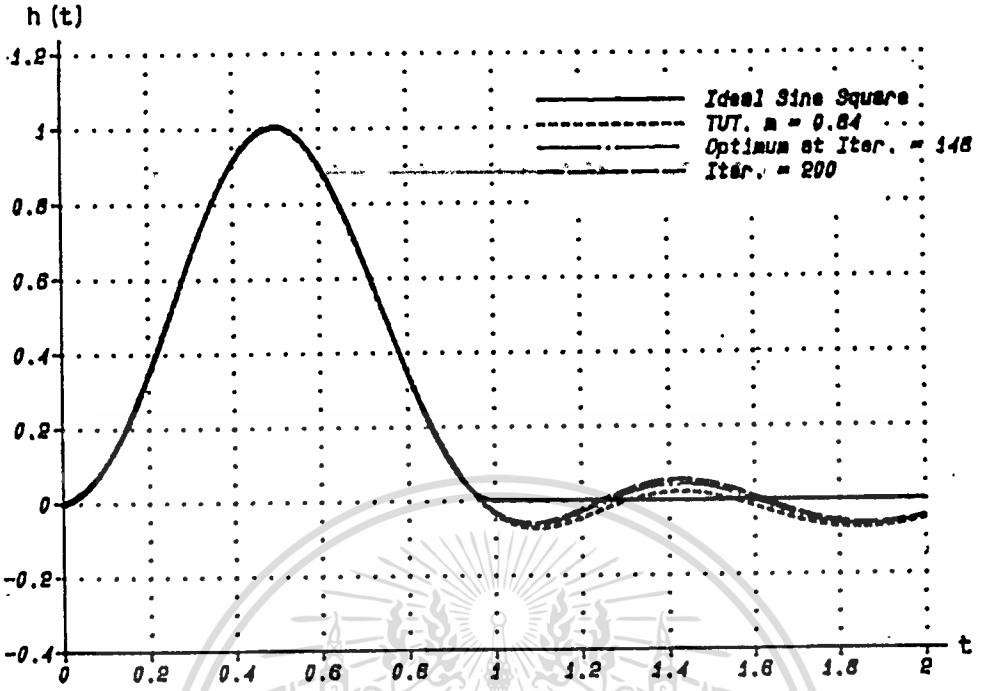
ตารางที่ 4.6 แสดงการคำนวณซ้ำด้วยวิธี คอนจุเกจ-เกรเดียนท์ ที่ k=0.04%



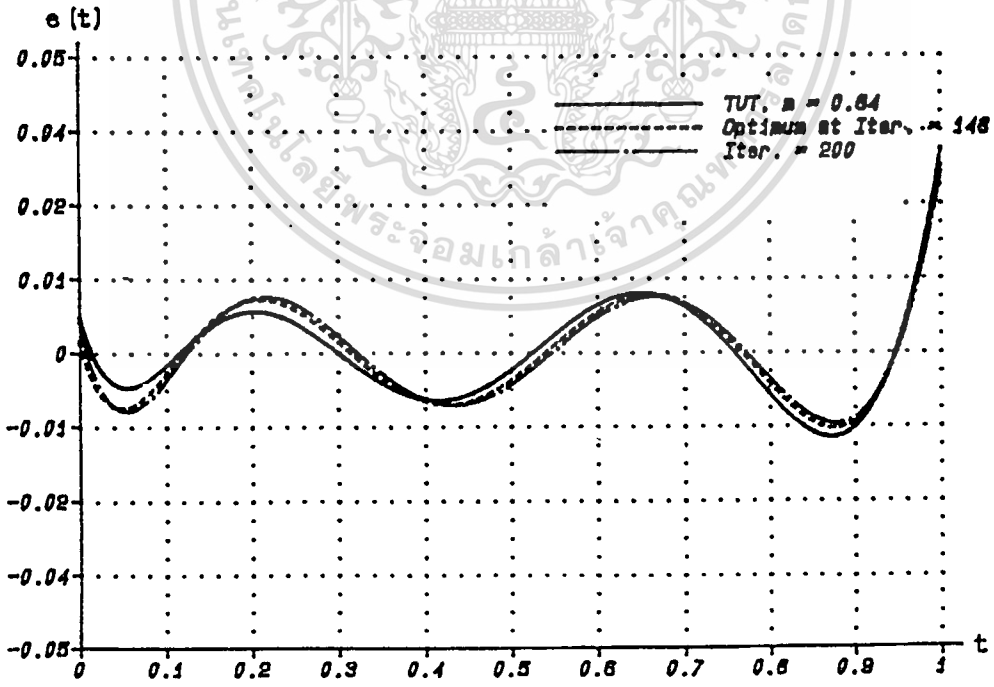

กราฟที่ 4.7 แสดงการคำนวณซ้ำด้วยวิธีสตีปเปส-ดีเซนที่ k=0.04%



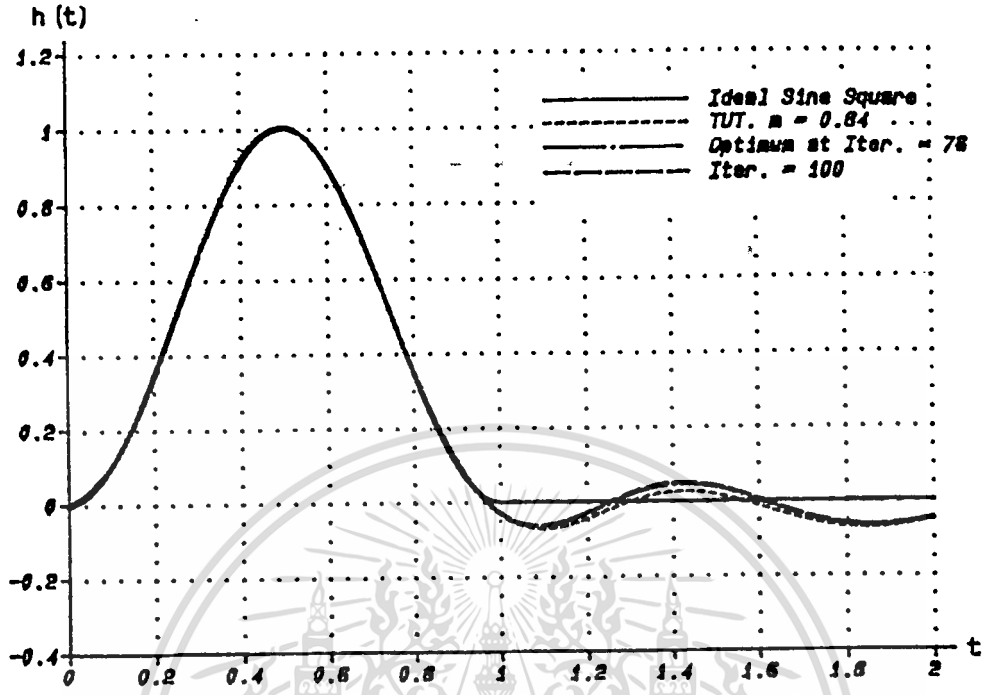

กราฟที่ 4.8 แสดงการคำนวณซ้ำด้วยวิธีคอนจุเกจ-เกรเดียนท์ที่ k=0.04%



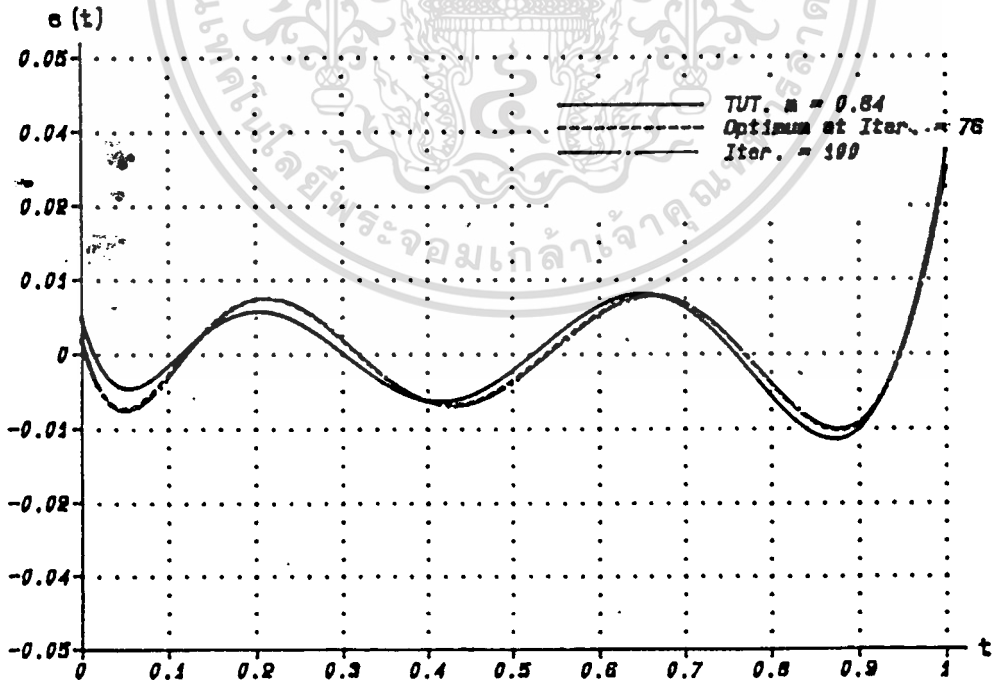

รูปที่ 4.13 แสดงผลตอบสนองจากการประมาณฟังก์ชั่นโดยวิธี สตีปเปส-ดีเซน




รูปที่ 4.14 แสดงค่าผิดพลาดจากการประมาณฟังก์ชั่นโดยวิธี สตีปเปส-ดีเซน




รูปที่ 4.15 แสดงผลตอบสนองจากการประมาณฟังก์ชั่นโดยวิธีคอนจุเกจ-เกรเดียนท์ที่ m=0.84,k=0.04%




รูปที่ 4.16 แสดงค่าผิดพลาดจากการประมาณฟังก์ชั่นโดยวิธีคอนจุเกจ-เกรเดียนท์ที่ m=0.84,k=0.04

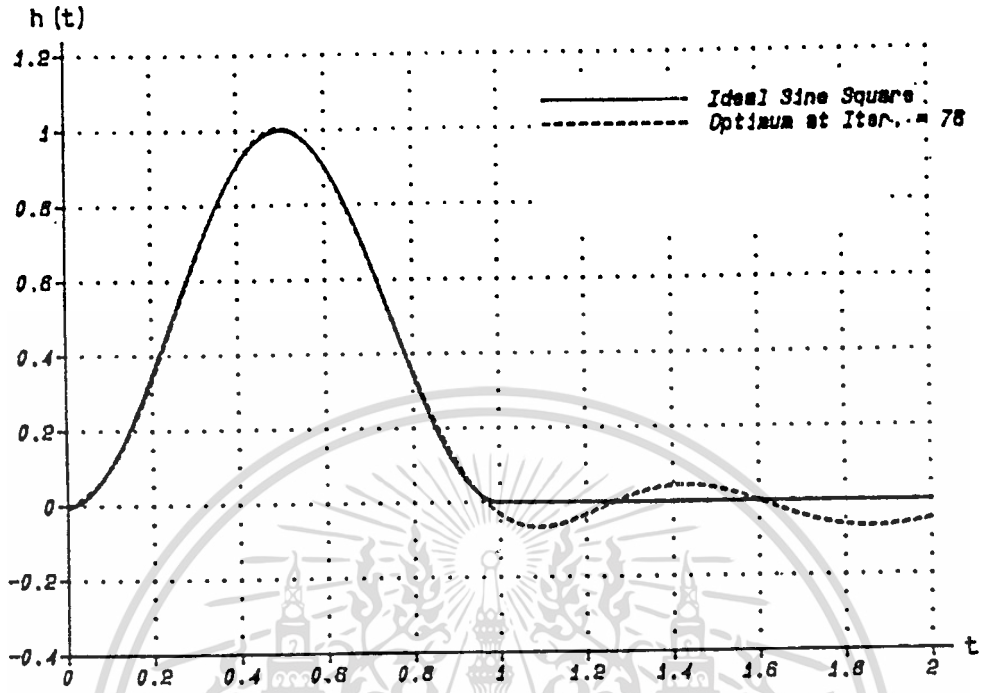
จากตารางที่ 4.6 ที่ Iter.=76 ของวิธี ดอนจุเกจ-เกรเดียนท์ จะให้ค่าผิดพลาดต่ำที่สุด แสดงว่าเป็นจุด Optimum เพราะฉะนั้นสามารถเขียนฟังก์ชั่น h(t) ของพัลซ์ชายน์กำลังสองที่ประมาณฟังก์ชั่นได้ดัง

$$h(t) = 0.1653e^{-2.3920t}\cos(0.7683t)+1.2283e^{-2.3920t}\sin(0.7683t) -0.1666e^{-1.5701t}\cos(2.2756t)-0.3579e^{-1.5701t}\sin(2.2756t) \quad ...(4.14)$$

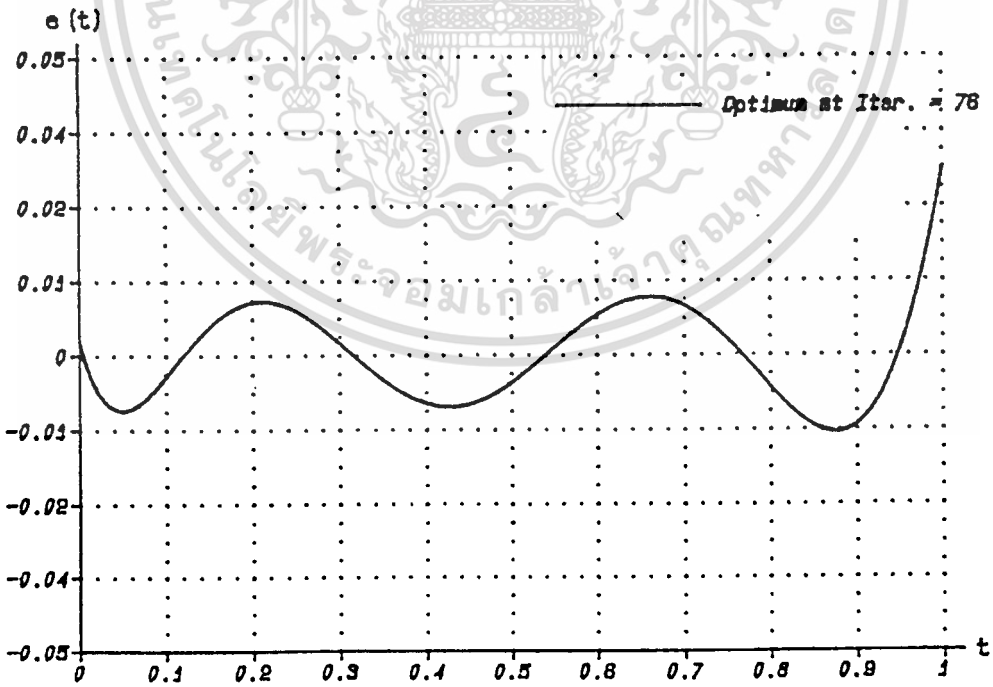
จากสมการ (4.14) ทำการ Take Laplace จะได้ทรานส์เฟอร์ฟังก์ชั่นดัง

$$H(S) = \frac{-0.0013S^3 - 0.0148S^2 -0.7306S + 3.4428}{S^4 + 7.9242S^3 + 28.9790S^2 + 56.3901S + 48.2501} \quad ...(4.15)$$

จากสมการที่ 4.15 สามารถนำทรานส์เฟอร์ฟังก์ชั่นที่ได้ไปสร้างวงจรเน็ทเวิกร์ แบบแอ็คทีฟเน็ทเวิกร์ได้ในบทต่อไป




รูปที่ 4.17 แสดงผลตอบสนองจากการประมาณฟังก์ชั่นด้วยกรรมวิธี Optimization




รูปที่ 4.18 แสดงค่าผิดพลาดจากการประมาณฟังก์ชั่นด้วยกรรมวิธี Optimization

# บทที่ 5
# การออกแบบวงจรแอ็คทีฟเน็ทเวิกร์

## 5.1 การออกแบบวงจรแอ็คทีฟเน็ทเวิกร์

ในการออกแบบวงจรเน็ทเวิกร์แบบ อาร์ซีแอ็คทีฟ (R-C Active Network) [6] จากทรานส์เฟอร์ฟังก์ชั่นหนึ่งๆสามารถทำได้ 2 วิธีคือ

1.**วิธีคาสเคด** (Cascade) โดยการแยกองค์ประกอบของเน็ทเวิกร์ฟังก์ชั่น ให้อยู่ในรูปผลคูณของเทอมอันดับที่หนึ่ง (First Order) หรือ เทอมอันดับที่สอง (Second Order) ในกรณีที่มีโพลจริงคูณอยู่ด้วย แต่ละเทอมสามารถนำมาสร้าง วงจร อาร์ซี แอ็คทีฟที่มีรูปแบบเดียวกัน แล้วนำมาอนุกรมกันจะได้วงจรรวมซึ่งแทนด้วยทรานส์เฟอร์ฟังก์ชั่นทั้งหมด ซึ่งมีข้อดีหลายประการคือ เนื่องจากวงจรแต่ละส่วนแทนด้วยเทอมอันดับสองเท่านั้น ทำให้รูปแบบวงจรไม่ซับซ้อน ใช้อุปกรณ์จำนวนน้อยและง่ายในการคำนวณ

2.**วิธีโดยตรง** (Direct) คือการสร้างวงจรแอ็คทีฟ อาร์ซี วงจรเดียวแทนด้วยทรานส์เฟอร์ฟังก์ชั่นทั้งหมด รูปแบบวงจรจริงซับซ้อนแตกต่างกันไปตามอันดับสูงสุดของเน็ทเวิกร์ฟังก์ชั่น

สำหรับวิทยานิพนธ์นี้เลือกใช้วิธีคาสเคด กำหนดให้ทรานส์เฟอร์ฟังก์ชั่นที่ใช้ในการออกแบบวงจรเน็ทเวิกร์ในรูปเศษและส่วน ดัง

$$H(s) = \frac{a_m s^m + a_{m-1} s^{m-1} + \ldots + a_0}{s^n + b_{n-1} s^{n-1} + \ldots + b_0} \qquad \ldots(5.1)$$

m เป็นอันดับของโพลิโนเมียลเศษ และ n เป็นอันดับของโพลิโนเมียลส่วน โดยที่ $n \geq m$

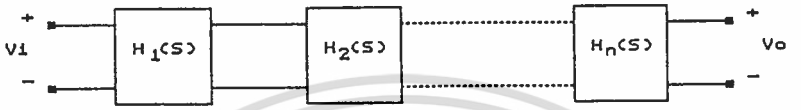
a และ b เป็นสัมประสิทธิ์ที่เป็นตัวเลขจำนวนจริง

s เป็นตัวแปรความถี่เชิงซ้อน (Complex-Frequency Variable)

เขียนทรานส์เฟอร์ฟังก์ชั่นในรูปของผลคูณได้

$$H(s) = H_1(s) \cdot H_2(s) \cdot \ldots \cdot H_n(s) \qquad \ldots(5.2)$$

จากทรานส์เฟอร์ฟังก์ชั่นในสมการ (5.2) สามารถกระทำในรูปของวงจรคาสเคดกัน โดยนำทรานส์เฟอร์ฟังก์ชั่นอันดับหนึ่งหรือทรานส์เฟอร์ฟังก์ชั่นอันดับสองมาต่ออนุกรมดังรูปที่ 5.1




$H(S) = H_1(S)H_2(S)...H_n(S) = Vo/Vi$

รูป 5.1 แสดงการต่ออนุกรม

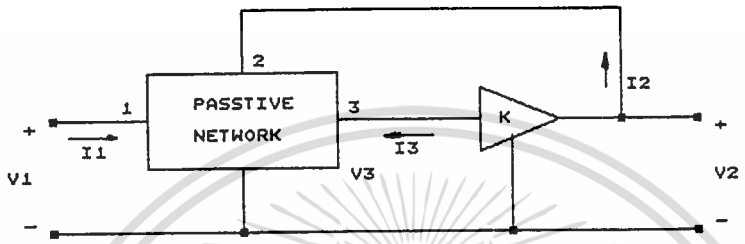
โดยทรานส์เฟอร์ฟังก์ชั่นอันดับสองทั่วๆไป มีสมการเป็น

$$H(s) = \frac{a_2s^2 + a_1s + a_0}{s^2 + b_1s + b_0} \qquad ...(5.3)$$

และทรานส์เฟอร์ฟังก์ชั่นอันดับหนึ่งมีสมการเป็น

$$H(s) = \frac{a_1s + a_0}{s + b_0} \qquad ...(5.4)$$

จากรูป 5.1 แต่ละองค์ประกอบของฟังก์ชั่นอยู่ในรูปผลคูณของเทอมอันดับหนึ่ง และเทอมอันดับสอง ซึ่งแต่ละเทอมสามารถนำมาสร้างวงจร เน็ทเวิกร์ แบบ อาร์ซี แอ็คทีฟได้โดยใช้อุปกรณ์แอ็คทีฟประเภท VCVS (Voltage Controled Voltage Source) ซึ่งในทางปฏิบัติก็คือวงจรขยายสัญญาณที่ใช้โอเปอร์เรชั่นแนลแอมปริฟายเออร์ (Operational Amplifier) ต่อร่วมกับวงจรแบบ พาสซีฟ (Passive Network) ซึ่งประกอบด้วยตัวต้านทาน (Resistor)และตัวเก็บประจุ(Capacitor) ดังรูปที่ 5.2 โดยที่ n เป็นอันดับที่ของฟังก์ชั่น อาจเป็นจำนวนคู่หรือจำนวนคี่ก็ได้ ถ้า n เป็นจำนวนคู่จะใช้วงจรอันดับสองทั้งหมดต่อคาสเคดกัน และถ้า n เป็นจำนวนคี่จะใช้วงจรอันดับสองคาสเคดกับวงจรอันดับหนึ่ง




รูป 5.2 แสดงวงจรเน็ทเวิกร์ 3 พอทร์ ต่อร่วมกับ ออฟแอมป์ (Op-Amp)

จากรูป 5.2 พิจารณาเน็ทเวิกร์ 3 พอทร์ (Three Port Network) ที่ต่อร่วมกับ VCVS โดยคุณสมบัติของ VCVS หาได้ดัง

$$I_3(s) = 0,\ V_2(s) = 0 \qquad \ldots(5.5)$$

จากรูป 5.2 สามารถหา โวลเตจทรานส์เฟอร์ฟังก์ชั่นได้ดัง

$$V_2(s) = -kV_{31}(s) \qquad \ldots(5.6)$$

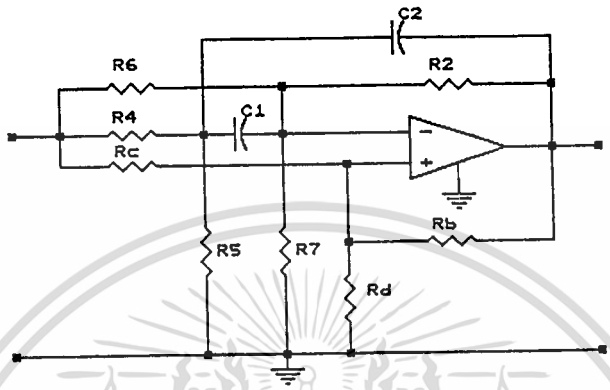
$$V_1(s) = V_{33}(s) + V_{32}(s) \qquad \ldots(5.7)$$

เพราะฉะนั้น

$$\frac{V_2(s)}{V_1(s)} = \frac{-kV_{31}(s)}{V_{33}(s) + V_{32}(s)} \qquad \ldots(5.8)$$

รูปแบบของการสร้างวงจรจากทรานส์เฟอร์ฟังก์ชั่นอันดับสอง มีหลายรูปแบบด้วยกัน สำหรับหัวข้อต่อไปนี้ จะกล่าวถึงรูปแบบ การสร้างวงจรสำหรับฟังก์ชั่นที่มีแต่โพลทั้งหมด (All Pole Function) และฟังก์ชั่นที่มีโพล (Pole) ร่วมกับซีโร่ (Zero) โดยใช้หลักการของแคนนอนิคัล (Single Amplifier Biquad - Canonical Technique) โดยใช้วงจรเฟรน (Friend Circuit)

## 5.1 วงจรเฟรนอันดับสอง




รูปที่ 5.3 แสดงรายละเอียดวงจรเฟรนอันดับสอง

จากรูปที่ 5.1 กำหนดให้โวลเตจทรานส์เฟอร์ฟังก์ชั่นเป็น

$$H(s) = \frac{a_2 s^2 + a_1 s + a_0}{s^2 + b_1 s + b_0} \qquad ...(5.9)$$

เมื่อ

$$G_x = 1/R_x \qquad ...(5.10)$$

$$a_2 = G_c/G_a \qquad ...(5.11)$$

$$a_1 = \frac{[C_1 G_c(G_1+G_2+G_3)+C_2 G_2(G_2+G_3)-(C_1 G_4(G_a+G_b)-(C_1+C_2)G_6(G_a+G_b)]}{C_1 C_2 G_a} \qquad ...(5.12)$$

$$a_0 = \frac{[G_1 G_c(G_2+G_3)-G_1 G_6(G_a+G_b)]}{C_1 C_2 G_a} \qquad ...(5.13)$$

$$b_1 = \frac{[(C_1+C_2)(G_aG_2-G_bG_3)-C_1G_1G_b]}{C_1C_2G_a} \qquad ...(5.14)$$

$$b_0 = \frac{[G_1(G_aG_2-G_bG_3]}{C_1C_2G_a} \qquad ...(5.15)$$

จากสมการ (5.11) ถึงสมการ (5.15) สามารถคำนวณหาค่า G ต่างๆได้ดังนี้

$$G_1 = G_2G_a/2G_b[-b_1+\sqrt{b_1^2+4b_0(1+C_1/C_2)G_b/G_a}] \qquad ...(5.16)$$

$$G_4 = G_1G_a/(G_a+G_b)[a_2+a_0(1+C_1/c_2)C_2^2/C_1^2 - a_1C_2/G_1] \qquad ...(5.17)$$

$$G_2 = C_1C_2b_0/G_1 + G_bG_3/G_a \qquad ...(5.18)$$

$$G_5 = G_1-G_4 \qquad ...(5.19)$$

$$G_6 = \alpha G_3 \qquad ...(5.20)$$

$$G_7 = G_3-G_6 \qquad ...(5.21)$$

$$G_c = a_2G_a \qquad ...(5.22)$$

$$G_d = G_a-G_c \qquad ...(5.23)$$

กำหนดให้ $\alpha$ ในวงจรเฟรน

$$\alpha = G_b/(G_6+G_7) \qquad ...(5.24)$$

หรือ

$\alpha = 0$ เมื่อ $a_0-a_2b_0 > 0$

หรือ

$\alpha = 1$ เมื่อ $a_0-a_2b_0 < 0$

ในกรณีที่ $a_1 < 0$ กำหนดให้โวลท์เตจทรานส์เฟอร์ฟังก์ชั่นเป็น

$$\beta H(s) = \frac{a_2's^2 + a_1's + a_0'}{s^2 + b_1s + b_0} \qquad ...(5.25)$$

เมื่อ

$$a_2' = \beta a_2$$
$$a_1' = \beta a_1$$
$$a_0' = \beta a_0$$
$$\beta < 1$$

จากทรานส์เฟอร์ฟังก์ชั่นในสมการ (4.15) สามารถกระทำการคาสเคสกันในลักษณะของวงจรฟิลเตอร์อันดับสองต่อร่วมกัน โดยแยกพิจารณาแยกวงจร เน็ทเวิกร์ แบบแอ็คทีฟ อันดับสองเป็น 2 วงจรดัง

$$H(s) = \frac{0.0013s^2 + 0.0203s + 0.8160}{s^2 + 4.7840s + 6.3120} \cdot \frac{-s + 4.2177}{s^2 + 43.1402s + 7.6436} \quad ...(5.26)$$

ทรานส์เฟอร์ฟังก์ชั่นที่ต้องการหาค่าพารามิเตอร์ โดยใช้ วงจรเฟรนอันดับสอง ให้ $C_1 = C_2 = 1$, $G_a = G_b = 1$

$$H(s) = \frac{0.0013s^2 + 0.0203s + 0.8160}{s^2 + 4.7840s + 6.3120} \quad ...(5.27)$$

เพราะฉะนั้นจะได้ค่าพารามิเตอร์ต่างๆคือ

$G_1 = 1.891$ $G_2 = 167.64$
$G_3 = 164.3$ $G_4 = 0.4225$
$G_5 = 1.4685$ $G_6 = 0$
$G_7 = 164.3$ $G_d = 0.9987$
$G_c = 0.0013$ $G_b = 1$

และค่าตัวต้านทานแต่ละตัวหาได้จากส่วนกลับของ Admittance

$R_2 = 0.006$ $R_4 = 2.3669$
$R_5 = 0.6809$ $R_7 = 0.0061$
$R_c = 769.23$ $R_d = 1.0013$
$R_b = 1$ $C_1 = C_2 = 1$

ในทำนองเดียวกัน ทรานส์เฟอร์ฟังก์ชั่นอันดับสองอีกส่วนหนึ่งของสมการ (5.26) สมการดังกล่าวสามารถหาค่าพารามิเตอร์จาก วงจรเฟรนอันดับสองให้ $C_1 = C_2 = 1, G_a = G_b = 1$

$$H(s) = \frac{-s + 4.2177}{s^2 + 3.1402s + 7.6436} \qquad ...(5.28)$$

เพราะฉะนั้นจะได้ค่าพารามิเตอร์ต่างๆคือ

$G_1 = 2.6433$ $G_2 = 2.8917$

$G_3 = 0$ $G_4 = 2.0957$

$G_5 = 0.5476$ $G_6 = 0$

$G_7 = 0$ $G_d = 1$

$G_c = 0.0013$ $G_b = 1$

และค่าตัวต้านทานแต่ละตัวหาได้จากส่วนกลับของ Admittance

$R_2 = 0.3458$ $R_4 = 0.4772$

$R_5 = 1.8262$ $R_d = 1$

$R_b = 1$ $C_1 = C_2 = 1$

## 5.2 การสเกล

โดยทั่วไปนิยมให้ความถี่คัดออฟ (Cut off Frequency) ของฟังก์ชั่นวงจรกรองความถี่เกิดขึ้นที่ $\omega=1$ และการคำนวณค่าอุปกรณ์ในวงจรกรองความถี่ก็นิยมกำหนดค่า R หรือ C มีค่าเท่ากับ 1 ฉะนั้นเมื่อต้องการนำฟังก์ชั่นไปใช้งานที่ความถี่ใด จึงต้องทำการสเกลทางความถี่ (Frequency Scaling) ไปที่ความถี่นั้น และเพื่อให้สามารถเลือกใช้อุปกรณ์ที่เหมาะสมในทางปฏิบัติ จะต้องทำการสเกลทางขนาด (Magnitude Scaling) อีกด้วย

### 5.2.1 การสเกลทางความถี่

การสเกลทางความถี่ (Frequency Scale) จะมีผลต่อการเปลี่ยนแปลงค่าของอุปกรณ์จำพวกรีแอคทีฟ (Reactive) เช่นอุปกรณ์ L และ C เท่านั้น เนื่องจากอิมพีแดนซ์ (Impedance) ของอุปกรณ์มีค่าแปรตามความถี่ พิจารณาค่าอิมพีแดนซ์ของอุปกรณ์ L

$$|Z_L| = Z_L \qquad ...(5.29)$$

เพื่อจะทำให้อิมพิแดนซ์คงที่ การเปลี่ยนแปลงความถี่ ω จะต้องถูกชดเชยด้วยค่าที่สอดคล้องกันในอุปกรณ์ L

$$|Z_L| = \omega_L = (k_f\omega)(1/k_f)L = (k_f\omega)L_{new} \qquad ...(5.30)$$

ในทำนองเดียวกันสำหรับอุปกรณ์ C

$$|Z_C| = 1/\omega_C = 1/(k_f\omega)(1/k_f)C = 1/(k_f\omega)C_{new} \qquad ...(5.31)$$

จากที่กล่าวมาสามารถพิสูจน์ได้ว่า ผลของการสเกลทางความถี่ด้วยตัวประกอบ $k_f$ ที่มีต่อค่าของอุปกรณ์พาสซีฟ (Passive) มีดังนี้

$$L_{new} = (1/k_f)L_{old} \qquad ...(5.32)$$

$$Cnew = (1/k_f)C_{old} \qquad ...(5.33)$$

$$R_{new} = R_{old} \qquad ...(5.34)$$

5.2.2 <u>การสเกลทางขนาด</u>

การสเกลทางขนาด (Magnitude Scale) คือการเปลี่ยนแปลงค่าอิมพิแดนซ์ของอุปกรณ์ทุกตัวในวงจรด้วยตัวประกอบ $k_m$ โดยที่อิมพิแดนซ์ ของอุปกรณ์พาสซีฟมีค่าดังนี้

$$Z_R = R, \ |Z_L| = \omega L, \ |Z_C| = 1/\omega C \qquad ...(5.35)$$

ทำการสเกลทางขนาด โดยคูณด้วยค่า $k_m$ จะได้

$$k_mZ_R = k_mR, \ k_m|Z_L| = k_m\omega L, \ k_m|Z_C| = 1/(\omega C/k_m) \qquad ...(5.36)$$

จากที่กล่าวมาสามารถสรุปผลของการสเกลทางขนาดด้วยตัวประกอบ $k_m$ ที่มีต่ออุปกรณ์พาสซีฟมีดังนี้

$$R_{new} = k_mR_{old} \qquad ...(5.37)$$

$$L_{new} = k_mL_{old} \qquad ...(5.38)$$

$$C_{new} = (1/k_m)C_{old} \qquad ...(5.39)$$

ผลจากการสเกลค่าอุปกรณ์ที่คำนวณได้ ในที่นี้เราสเกลค่าอุปกรณ์ไปที่ความถี่ 10kHz เพราะฉะนั้นค่าอุปกรณ์ที่สเกลได้คือ

จากวงจรเฟรนดังสมการ (5.32) ได้ค่าอุปกรณ์ใหม่คือ

$R_2$ = 60 ohm $R_4$ = 23.669 Kohm

$R_5$ = 6.8 Kohm $R_7$ = 61 ohm

$R_c$ = 7.69 Mohm $R_d$ = 10 Kohm

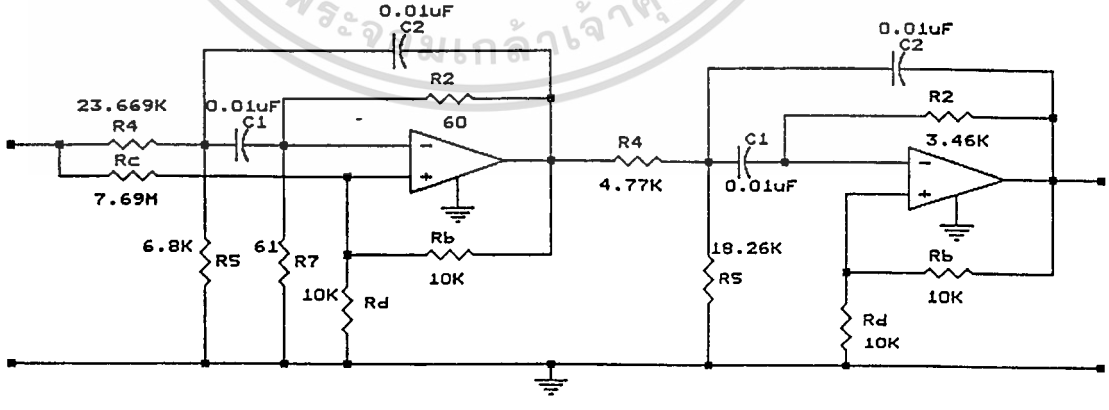
$R_b$ = 10 Kohm $C_1 = C_2$ = 0.01 uF

ในทำนองเดียวกัน จากสมการเฟรนดังสมการ (5.33) ได้ค่าอุปกรณ์ใหม่คือ

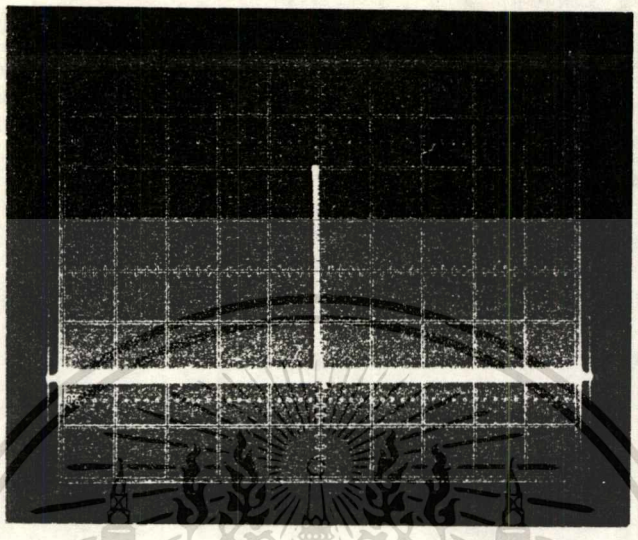
$R_2$ = 3.46 Kohm $R_4$ = 4.771 Kohm

$R_5$ = 18.26 Kohm $R_d$ = 1 Kohm

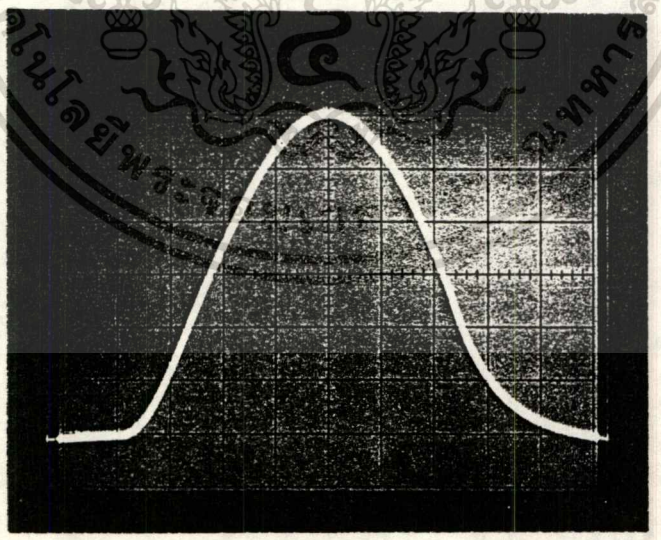
$R_b$ = 10 Kohm $C_1 = C_2$ = 1 uF




รูปที่ 5.5 แสดงวงจรใช้งานจริง



รูปที่ 5.6 สัญญาณอิมพัลซ์ที่ป้อนเข้าทางอินพุทของวงจรทดลอง
Volt/Division = 1 V , Time Sweep = 0.01 msec



รูปที่ 5.7 สัญญาณพัลซ์ซายน์กำลังสองจากวงจรทดลอง
Volt/Division = 1 V , Time Sweep = 1 msec

# บทที่ 6
# สรุปผลการวิจัย

การประมาณฟังก์ชั่นในโดเมนเวลาโดยการกำหนดโพลเริ่มต้นด้วยวิธีทรานส์ชิทชั่นนัล อุลตร้าสเฟียริคัล-ทอมป์สัน โพลีโนเมียล (TUT) นั้น โดยใช้ค่าพารามิเตอร์ m = 0.84 จะให้ผลตอบสนองต่อสัญญาณพัลซ์ชายน์กำลังสองที่ดี ซึ่งผลตอบสนองจะอยู่ใกล้จุด Optimum หรือเรียกว่า Local Optimum และเมื่อนำค่าโพลกับค่าสัมประสิทธิ์เหล่านั้นมาเป็นค่าเริ่มต้น สำหรับกรรมวิธี Optimization ด้วยวิธี เน็คกาทีฟ-เกรเดียนท์ โดยใช้ทิศทางของ สตีปเปส-ดีเซน และทิศทางคอนจุเกจ-เกรเดียนท์ในการหาผลตอบสนองที่ดีที่สุดต่อไป จนกระทั่งได้จุด Optimum จริงๆ หรือเรียกว่า Global Optimum ซึ่งทิศทางสตีปเปส-ดีเซนจะให้จำนวนครั้งในการคำนวณซ้ำมากกว่าใช้ทิศทางของคอนจุเกจ-เกรเดียนท์ที่ให้ผลการคำนวณที่เร็วกว่ามี Rate Convergence ที่ดีกว่า และหาจุด Optimum ที่ดีกว่าด้วย ทำให้มีประสิทธิภาพในกรรมวิธี Optimization เพื่อหาผลตอบสนองต่อสัญญาณพัลซ์ชายน์กำลังสองที่ใกล้เคียงสัญญาณพัลซ์ชายน์กำลังสองในอุดมคติ และสามารถนำทรานส์เฟอร์ฟังก์ชั่นที่ได้มาประกอบเป็นวงจรเน็ทเวิกร์ เพื่อสร้างสัญญาณพัลซ์ชายน์กำลังสองให้ได้ใกล้เคียงกับการสังเคราะห์

การประมาณฟังก์ชั่นในโดเมนเวลาและกรรมวิธี Optimization ของการวิจัยในวิทยานิพนธ์นี้ อาจจะเป็นแนวทางสำหรับผู้สนใจทั่วๆไป ไม่ว่าจะใช้ในทางวิศวกรรม หรือในแขนงวิชาอื่นๆ ก็สามารถนำเทคนิคต่างๆไปประกอบได้

## เอกสารอ้างอิง


1. ธนิตพงษ์ วิบูลยานนท์,ชวลิต เบญจางคประเสริฐ,ไพศาล สิทธิโยภาสกุล,อรลาภ แสงอรุณ, ดร.กนก เจนจิระพงศ์เวช,"การประมาณฟังก์ชั่นซายน์กำลังสองโดยวิธีเน็คกาทีฟ-เกรเดียนท์ด้วยการกำหนดโพลเริ่มต้น" การประชุมทางวิชาการวิศวกรรมไฟฟ้า 9 สถาบัน ครั้งที่ 14 มหาวิทยาลัยสงขลานครินทร์ หาดใหญ่ หน้า 5-66 ถึง 5-71 วันที่ 7-8 พ.ย.2534
2. ไชยศักดิ์ แซ่ว่อง, กนก เจนจิระพงศ์เวช,"วงจรกรองความถี่แบบบัตเตอร์เวิธ-อุลตร้าสเฟียริคัลโพลิโนเมียล" การประชุมทางวิชาการวิศวกรรมไฟฟ้า 9 สถาบัน ครั้งที่ 10 จุฬาลงกรณ์มหาวิทยาลัย เล่มที่ 2 หน้า 239-247 พฤศจิกายน 2530
3. วิวัฒน์ กิรานนท์ โมไนย ไกรฤกษ์ "ทฤษฎีโครงข่ายวงจรไฟฟ้า" วศ.สจล. 142 ภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สจล.
4. G.C.Temes and S.K.Mitra, "Modern Filter Theory and Design",John Wiley & Sons,Inc., 1973.
5. A.Budak, "Passive and Active Network Analysis and Synthesis", Houghton Mifflin Company, 1974
6. Harry Y.F. LAM, "Analog and Digital Filters Design and Realization" Prentice-Hall, INC., Englewood Cliffs,New Jersey 07632
7. V.Valkenburg, "Analog Filter Design", Holt Saunders,1982.
8. R.Kennedy,"Sine Squared Pulse in Television System Analysis",RCA Review, Vol.21,No.2,p.253, June 1960
9. C.A.Siocos, "Chrominance-to-Luminance Ratio and Timing Measurements",IEEE Trans.,BC-14,1,pp.1-4, March 1968.
10. E.D.Rainville, "Special Functions", The Macmillan Company, pp.254-283, 1976
11. Thomson W.E.,"The Synthesis of Network to have a Sine-Squared Impulse Response", Proc I.E.E.(LONDON),pt. Vol.99, pp.373, 1952
12. Kendall L.SU.,"Time Domain Synthesis of Linear Network", Englewood Cliffe, N.E., Prentice-Hall, pp. 104-106, 1971
13. Johnson, D.E. and Johnson, J.R. "Low-pass Filters using Ultraspherical Polynomial", IEEE Trans. on circuit Theory, Vol CT-13, No.3, pp. 364-368, December 1966
14. Pelless, Y. and Murakami,T.,"Analysis And Synthesis Of Transitional Butterworth Filters And Bandpass Ampliers", RCA Review, Vol.18,

pp.60-94, March. 1957

15. Aiell, G.L. and Anagelo, P.M. "Transitional Legendre Thomson Filter",IEEE Trans. On Circuits And System", Vol. CAS-12 pp.159-162, March. 1974

16. Alkis Constantinides, "Applied Numerical Method With Personal Computers" (McHill)

17. Thomas R.Cuthbert,JR.,"Optimization Using Personal Computers With Application To Electrical Networks"

18. Kendall E.Atkinson, "An Introduction To Numerical Analysis",Second Edition, University Of Iowa

19. David M.Himmelblaw, "Applied Nonlinear Programming" ,The University Of Texas,Austin,Texas

20. Jasbir S.Arora,"Introduction To Optimum Design" McGraw-Hill Book Company.Inc, 1989

21. S.S.Rao, "Optimization Theory and Application Indian Institute of Technology", Kanpur, 1968

# ภาคผนวก
# รายละเอียดโปรแกรม

```
PROGRAM Conjugate_Gradient_Method;


{$I ega.INC}
const ArraySize = 50;        { Size of the matrix }

type vector = array[1..ArraySize] of real;
     matrix = array[1..ArraySize] of vector;

var Dimen,Poles,Realpoles,col,iter
                          : integer;   { Dimen of the square matrix }
    Coefficients,Pole     : matrix;    { The matrix }
    Constants             : vector;    { Constant terms in the equations }
    Solution              : vector;    { Solution to the set of equations }
    Error                 : byte;      { Flags if something went wrong }

var LowerLimit,UpperLimit  : real;      { Limits of integration }
    NumIntervals           : integer;   { Number of intervals }
    Integral               : real;      { Value of the integral }
    A1,A2,B1,B2,dmax,kmax1,kmax2,kmax3,kmax4,kmax5,kmax6,kmax7,kmax8 : real;
    a11,b11,b22,a12,a13,a14,a21,a22,a23,a24,a31,a32,a33,a34,a41,a42,a43,a44,
    c11,c12,c13,c14,dE1,dE2,dE3,dE4,dE5,dE6,dE7,dE8        : real;
    pole1,pole2,pole3,pole4,Opole1,Opole2,Opole3,Opole4,OA1,OA2,OB1,OB2 : real;
    chr,check,ch       : char;
    magnitude,rate     : real;
    magnitude4,magnitude5                          : real;
    conjugate1,conjugate2,conjugate3,conjugate4    : real;
    direction1,direction2,direction3,direction4    : real;

{$I PLOT.INC}
{$I INIT.INC}
{$I ERROR.INC}

begin { program Conjugate Grendient }

dimen := 4;
clrscr;
{use pole at TUT m =0.84 }
  pole1 := -2.2302;
  pole2 := 0.7961;
  pole3 := -1.5701;
  pole4 := 2.2760;
  A1 := 0.1565;
  B1 := 1.1339;
  A2 := -0.1596;
  B2 := -0.3463;

  GetData(LowerLimit,UpperLimit,NumIntervals);
  iter := 0;
  SimpsonErr(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[6,1] := Integral;
OA1:=A1;OB1:=B1;OA2:=A2;OB2:=B2;
Opole1:=Pole1;Opole2:=Pole2;Opole3:=Pole3;Opole4:=Pole4;
clrscr;
  writeln ('These program find next coefficients value of FILTER');
  writeln ('Pole1-2 = ',Opole1:6:4,' ± j',Opole2:6:4);
  writeln ('Pole3-4 = ',Opole3:6:4,' ± j',Opole4:6:4);
  writeln ('A1 = ',OA1:6:4,'  B1 = ',OB1:6:4);
  writeln ('A2 = ',OA2:6:4,'  B2 = ',OB2:6:4);
  writeln ('Iteration number : ',iter,'     ERROR = ',coefficients[6,1]:10:9);
  writeln;
  lowvideo;
  write ('                                Please wait');
  normvideo;
  Simpsonp5(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,1] := Integral;
```

```
  Simpsonp6(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,2] := Integral;
  Simpsonp7(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,3] := Integral;
  Simpsonp8(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,4] := Integral;

  coefficients[7,1] := coefficients[5,1];
  coefficients[7,2] := coefficients[5,2];
  coefficients[7,3] := coefficients[5,3];
  coefficients[7,4] := coefficients[5,4];


  magnitude := (sqrt(sqr(Abs(coefficients[5,1])) +
                     (sqr(Abs(coefficients[5,2])) +
                     (sqr(Abs(coefficients[5,3])) +
                     (sqr(Abs(coefficients[5,4])))));

  rate      := 0.001;

  kmax5 := (rate*Abs(coefficients[5,1]))/magnitude;
  kmax6 := (rate*Abs(coefficients[5,2]))/magnitude;
  kmax7 := (rate*Abs(coefficients[5,3]))/magnitude;
  kmax8 := (rate*Abs(coefficients[5,4]))/magnitude;

pole[1,1] := (pole1)-(kmax5*coefficients[5,1]);
pole[1,2] := (pole2)-(kmax6*coefficients[5,2]);
pole[2,1] := (pole3)-(kmax7*coefficients[5,3]);
pole[2,2] := (pole4)-(kmax8*coefficients[5,4]);

  Simpson1(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[1,1] := Integral;
  Simpson2(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[1,2] := Integral;
  Simpson3(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[1,3] := Integral;
  Simpson4(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[1,4] := Integral;
  Simpson5(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[2,1] := Integral;
  Simpson6(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[2,2] := Integral;
  Simpson7(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[2,3] := Integral;
  Simpson8(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[2,4] := Integral;
  Simpson9(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[3,1] := Integral;
  Simpson10(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[3,2] := Integral;
  Simpson11(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[3,3] := Integral;
  Simpson12(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[3,4] := Integral;
  Simpson13(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[4,1] := Integral;
  Simpson14(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[4,2] := Integral;
  Simpson15(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[4,3] := Integral;
  Simpson16(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[4,4] := Integral;
{writeln('CONSTANT');}
  Simpson17(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  constants[1] := Integral;
  Simpson18(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
```

```
  constants[2] := Integral;
  Simpson19(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  constants[3] := Integral;
  Simpson20(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  constants[4] := Integral;
  {begin} { Gaussian_Elimination }
{  GetData(Dimen,Coefficients,Constants);}
  Gaussian_Elimination(Dimen,Coefficients,Constants,Solution,Error);
  Resultg(Dimen,Coefficients,Constants,Solution,Error);

{end.} { Gaussian_Elimination }
  Iter  := 1;

  coefficients[4,1] := coefficients[7,1];
  coefficients[4,2] := coefficients[7,2];
  coefficients[4,3] := coefficients[7,3];
  coefficients[4,4] := coefficients[7,4];

  pole1 := pole[1,1];
  pole2 := pole[1,2];
  pole3 := pole[2,1];
  pole4 := pole[2,2];

  SimpsonErr(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[6,1] := Integral;
write ('    new pole1 = ',pole[1,1]:6:5);writeln ('      new pole2 = ',pole[1,2]:6:5);
write ('    new pole3 = ',pole[2,1]:6:5);writeln ('      new pole4 = ',pole[2,2]:6:5);
writeln;
write('    Gradient1 = ',coefficients[5,1]:6:5);writeln ('      Gradient2 = ',coefficients[5,2]:6:5);
write('    Gradient3 = ',coefficients[5,3]:6:5);writeln ('      Gradient4 = ',coefficients[5,4]:6:5);
writeln;
write  ('       % Rate = ',rate:6:5);  writeln ('     Iteration num = ',iter);
writeln('       Error = ',coefficients[6,1]:10:9);
write('Step size 1-4 = ',kmax5:6:5,'  ',kmax6:6:5,'  ',kmax7:6:5,'  ',kmax8:6:5);
writeln;
writeln ('Press< Enter >to Next Iter, <1>Plot sine^2, <2>Plot Error, <0>to Exit');
delay(300);
while coefficients[6,1] <> 0  do
begin
Repeat
clrscr;
  writeln ('These program find next coefficients value of FILTER');
  writeln ('Pole1-2 = ',pole1:6:4,' ± j',pole2:6:4);
  writeln ('Pole3-4 = ',pole3:6:4,' ± j',pole4:6:4);
  writeln ('A1 = ',A1:6:4,'  B1 = ',B1:6:4);
  writeln ('A2 = ',A2:6:4,'  B2 = ',B2:6:4);
  writeln ('Iteration number : ',iter,'     ERROR = ',coefficients[6,1]:10:9);
  writeln;
  lowvideo;
  write ('                                  Please wait');
  normvideo;
  Simpsonp5(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,1] := Integral;
  Simpsonp6(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,2] := Integral;
  Simpsonp7(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,3] := Integral;
  Simpsonp8(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,4] := Integral;

  magnitude5 := (sqrt(sqr(Abs(coefficients[5,1]))) +
                 (sqr(Abs(coefficients[5,2]))) +
                 (sqr(Abs(coefficients[5,3]))) +
                 (sqr(Abs(coefficients[5,4]))));
  magnitude4 := (sqrt(sqr(Abs(coefficients[4,1]))) +
```

```
                (sqr(Abs(coefficients[4,2]))) +
                   (sqr(Abs(coefficients[4,3]))) +
                   (sqr(Abs(coefficients[4,4])));

conjugate1 := (sqr(magnitude5/magnitude4))*coefficients[4,1];
conjugate2 := (sqr(magnitude5/magnitude4))*coefficients[4,2];
conjugate3 := (sqr(magnitude5/magnitude4))*coefficients[4,3];
conjugate4 := (sqr(magnitude5/magnitude4))*coefficients[4,4];

direction1 := coefficients[5,1] + conjugate1;
direction2 := coefficients[5,2] + conjugate2;
direction3 := coefficients[5,3] + conjugate3;
direction4 := coefficients[5,4] + conjugate4;

kmax5 := (rate*Abs(coefficients[5,1]))/magnitude5;
kmax6 := (rate*Abs(coefficients[5,2]))/magnitude5;
kmax7 := (rate*Abs(coefficients[5,3]))/magnitude5;
kmax8 := (rate*Abs(coefficients[5,4]))/magnitude5;

pole[1,1] := (pole1)-(kmax5*direction1);
pole[1,2] := (pole2)-(kmax6*direction2);
pole[2,1] := (pole3)-(kmax7*direction3);
pole[2,2] := (pole4)-(kmax8*direction4);

Simpson1(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[1,1] := Integral;
Simpson2(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[1,2] := Integral;
Simpson3(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[1,3] := Integral;
Simpson4(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[1,4] := Integral;
Simpson5(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[2,1] := Integral;
Simpson6(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[2,2] := Integral;
Simpson7(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[2,3] := Integral;
Simpson8(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[2,4] := Integral;
Simpson9(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[3,1] := Integral;
Simpson10(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[3,2] := Integral;
Simpson11(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[3,3] := Integral;
Simpson12(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[3,4] := Integral;
Simpson13(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[4,1] := Integral;
Simpson14(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[4,2] := Integral;
Simpson15(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[4,3] := Integral;
Simpson16(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
coefficients[4,4] := Integral;
{writeln('CONSTANT');}
Simpson17(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
constants[1] := Integral;
Simpson18(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
constants[2] := Integral;
Simpson19(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
constants[3] := Integral;
Simpson20(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
constants[4] := Integral;
{begin} { Gaussian_Elimination }
```

```
{  GetData(Dimen,Coefficients,Constants);}
  Gaussian_Elimination(Dimen,Coefficients,Constants,Solution,Error);
  Resultg(Dimen,Coefficients,Constants,Solution,Error);

{end.} { Gaussian_Elimination }
  pole1 := pole[1,1];
  pole2 := pole[1,2];
  pole3 := pole[2,1];
  pole4 := pole[2,2];

  coefficients[4,1] := coefficients[5,1];
  coefficients[4,2] := coefficients[5,2];
  coefficients[4,3] := coefficients[5,3];
  coefficients[4,4] := coefficients[5,4];

iter := iter + 1;

  SimpsonErr(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[6,1] := Integral;
write ('    new pole1 = ',pole[1,1]:6:5);writeln ('      new pole2 = ',pole[1,2]:6:5);
write ('    new pole3 = ',pole[2,1]:6:5);writeln ('      new pole4 = ',pole[2,2]:6:5);
writeln;
writeln('Conjugate1-4 = ',conjugate1:6:5,'  ',conjugate2:6:5,'  ',conjugate3:6:5,'  ',conjugate4:6:5);
writeln('Direction1-4 = ',direction1:6:5,'  ',direction2:6:5,'  ',direction3:6:5,'  ',direction4:6:5);
writeln;
write  ('       % Rate = ',rate:6:5);  writeln ('     Iteration num = ',iter);
writeln('        Error = ',coefficients[6,1]:10:9);
writeln('Step size 1-4 = ',kmax5:6:5,'  ',kmax6:6:5,'  ',kmax7:6:5,'  ',kmax8:6:5);
writeln;
writeln ('Press< Enter >to Next Iter, <1>Plot sine^2, <2>Plot Error, <0>to Exit');
delay(300);
A1:=Solution[1];
B1:=Solution[2];
A2:=Solution[3];
B2:=Solution[4];
  pole1 := pole[1,1];
  pole2 := pole[1,2];
  pole3 := pole[2,1];
  pole4 := pole[2,2];

Until  keypressed or (iter = 200);
Readln(ch);
If ch = '0' then exit;
If ch = '1' then
plot_2d_function_Sine;
If ch = '2' then
plot_2d_function_Error;

end;

end. { Main program }
```

```
PROGRAM STEEPEST_DESCENT;
{Adjust poles by using steepest decent and find coefficients
 by using least square}
{$I EGA.INC}
const ArraySize = 50;        { Size of the matrix }

type vector = array[1..ArraySize] of real;
     matrix = array[1..ArraySize] of vector;

var Dimen,Poles,Realpoles,col,iter
                        : integer;   { Dimen of the square matrix }
    Coefficients,Pole   : matrix;    { The matrix }
    Constants           : vector;    { Constant terms in the equations }
    Solution            : vector;    { Solution to the set of equations }
    Error               : byte;      { Flags if something went wrong }

var LowerLimit,UpperLimit  : real;      { Limits of integration }
    NumIntervals           : integer;   { Number of intervals }
    Integral               : real;      { Value of the integral }
    A1,A2,B1,B2,dmax,kmax1,kmax2,kmax3,kmax4,kmax5,kmax6,kmax7,kmax8 : real;
    a11,b11,b22,a12,a13,a14,a21,a22,a23,a24,a31,a32,a33,a34,a41,a42,a43,a44,
    c11,c12,c13,c14,dE1,dE2,dE3,dE4,dE5,dE6,dE7,dE8        : real;
    pole1,pole2,pole3,pole4,Opole1,Opole2,Opole3,Opole4,OA1,OA2,OB1,OB2 : real;
    chr,check,ch       : char;
    magnitude,rate     : real;

{$I PLOT.INC}
{$I INIT.INC}
{$i ERROR.INC}

begin { program Steepest descent }
dimen := 4;
clrscr;
{use pole at TUT m =0.84 }
  pole1 := -2.2302;
  pole2 := 0.7961;
  pole3 := -1.5701;
  pole4 := 2.2760;
  A1 := 0.1565;
  B1 := 1.1339;
  A2 := -0.1596;
  B2 := -0.3463;
  GetData(LowerLimit,UpperLimit,NumIntervals);
  iter := 0;
  SimpsonErr(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[6,1] := Integral;
while coefficients[6,1] <> 0  do
begin
Repeat
OA1:=A1;OB1:=B1;OA2:=A2;OB2:=B2;
Opole1:=Pole1;Opole2:=Pole2;Opole3:=Pole3;Opole4:=Pole4;
clrscr;
  writeln ('These program find next coefficients value of FILTER');
  writeln ('Pole1-2 = ',Opole1:6:4,' ± j',Opole2:6:4);
  writeln ('Pole3-4 = ',Opole3:6:4,' ± j',Opole4:6:4);
  writeln ('A1 = ',OA1:6:4,'  B1 = ',OB1:6:4);
  writeln ('A2 = ',OA2:6:4,'  B2 = ',OB2:6:4);
  writeln ('Iteration number : ',iter,'     ERROR = ',coefficients[6,1]:10:9);
  writeln;
  lowvideo;
  write ('                                  Please wait');
  normvideo;
  Simpsonp5(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,1] := Integral;
  Simpsonp6(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[5,2] := Integral;
```

```
    Simpsonp7(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[5,3] := Integral;
    Simpsonp8(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[5,4] := Integral;

    magnitude := (sqrt(sqr(Abs(coefficients[5,1])) +
                       (sqr(Abs(coefficients[5,2])) +
                       (sqr(Abs(coefficients[5,3])) +
                       (sqr(Abs(coefficients[5,4])))));

    rate      := 0.0004;

    kmax5 := (rate*Abs(coefficients[5,1]))/magnitude;
    kmax6 := (rate*Abs(coefficients[5,2]))/magnitude;
    kmax7 := (rate*Abs(coefficients[5,3]))/magnitude;
    kmax8 := (rate*Abs(coefficients[5,4]))/magnitude;

  pole[1,1] := (pole1)-(kmax5*coefficients[5,1]);
  pole[1,2] := (pole2)-(kmax6*coefficients[5,2]);
  pole[2,1] := (pole3)-(kmax7*coefficients[5,3]);
  pole[2,2] := (pole4)-(kmax8*coefficients[5,4]);

    Simpson1(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[1,1] := Integral;
    Simpson2(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[1,2] := Integral;
    Simpson3(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[1,3] := Integral;
    Simpson4(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[1,4] := Integral;
    Simpson5(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[2,1] := Integral;
    Simpson6(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[2,2] := Integral;
    Simpson7(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[2,3] := Integral;
    Simpson8(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[2,4] := Integral;
    Simpson9(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[3,1] := Integral;
    Simpson10(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[3,2] := Integral;
    Simpson11(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[3,3] := Integral;
    Simpson12(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[3,4] := Integral;
    Simpson13(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[4,1] := Integral;
    Simpson14(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[4,2] := Integral;
    Simpson15(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[4,3] := Integral;
    Simpson16(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    coefficients[4,4] := Integral;
  {writeln('CONSTANT');}
    Simpson17(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    constants[1] := Integral;
    Simpson18(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    constants[2] := Integral;
    Simpson19(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    constants[3] := Integral;
    Simpson20(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
    constants[4] := Integral;
    {begin} { Gaussian_Elimination } '
  {  GetData(Dimen,Coefficients,Constants);}
    Gaussian_Elimination(Dimen,Coefficients,Constants,Solution,Error);
```

```
  Resultg(Dimen,Coefficients,Constants,Solution,Error);

{end.} { Gaussian_Elimination }
  pole1 := pole[1,1];
  pole2 := pole[1,2];
  pole3 := pole[2,1];
  pole4 := pole[2,2];

iter := iter + 1;

  SimpsonErr(LowerLimit,UpperLimit,NumIntervals,Integral,Error);
  coefficients[6,1] := Integral;
write ('    new pole1 = ',pole[1,1]:6:5);writeln ('      new pole2 = ',pole[1,2]:6:5);
write ('    new pole3 = ',pole[2,1]:6:5);writeln ('      new pole4 = ',pole[2,2]:6:5);
writeln;
write('    Gradient1 = ',coefficients[5,1]:6:5);writeln ('       Gradient2 = ',coefficients[5,2]:6:5);
write('    Gradient3 = ',coefficients[5,3]:6:5);writeln ('      Gradient4 = ',coefficients[5,4]:6:5);
writeln;
write  ('       % Rate = ',rate:6:5);  writeln ('     Iteration num = ',iter);
writeln('        Error = ',coefficients[6,1]:10:9);
write('Step size 1-4 = ',kmax5:6:5,'  ',kmax6:6:5,'  ',kmax7:6:5,'  ',kmax8:6:5);
writeln;
writeln ('Press< Enter >to Next Iter, <1>Plot sine^2, <2>Plot Error, <0>to Exit');
delay(300);
Until  keypressed or (iter = 300);
Readln(ch);
If ch = '0' then exit;
If ch = '1' then
begin
plot_2d_function_Sine;
end;
If ch = '2' then
begin
plot_2d_function_Error;
end;
A1:=Solution[1];
B1:=Solution[2];
A2:=Solution[3];
B2:=Solution[4];
  pole1 := pole[1,1];
  pole2 := pole[1,2];
  pole3 := pole[2,1];
  pole4 := pole[2,2];
end;

  end. { Main program }
```

```
procedure Plot_2D_Function_sine;

 function log(x:real):real;
   begin
    log := ln(x) / ln(10);
   end;

 function fact(n:integer):real;
  var x  : real;
   begin
    x := 1;
    for n := n downto 1 do
     x := x*n;
    fact := x;
   end;

 function power(x,n:real):real;    { function power(base,order) }
   begin
    if x=0 then power := 0
    else
     if x>0 then  power := exp(ln(x)*n)
     else
      if frac(n)=0 then
       begin
        if odd(trunc(n)) then power := -exp(ln(abs(x))*n)
        else power := exp(ln(abs(x))*n);
       end
      else
       begin
        writeln('*** error ***');
        read;
       end;
   end;

var Sx,Sy                      : array[0..30] of real;
    X_min,X_max,Y_min,Y_max,
    AxGlb,AyGlb,BxGlb,ByGlb    : real;
    x1,y1                      : integer;
    LineStyle,dot              : byte;
    num                        : char;
    logs                       : boolean;

procedure DefineWorld(X1W,Y2W,X2W,Y1W: real);
 var X_1,Y_1,X_2,Y_2  : integer;
  begin
   X_min := X1W;
   Y_min := Y2W;
   X_max := X2W;
   Y_max := Y1W;
   X_1 := 3;
   Y_1 := 0;
   X_2 := XScreenMax-1;
   Y_2 := YScreenMax-4;
   BxGlb := (X_2-X_1)/(X2W-X1W);
   ByGlb := (Y_2-Y_1)/(Y2W-Y1W);
   AxGlb := X_1-X1W*BxGlb;
   AyGlb := Y_1-Y1W*ByGlb;
  end;

procedure SetLineStyle(m: byte);
  begin
   LineStyle := m;
   dot := 0;
  end;
```

```
procedure pset(x,y: real);
  begin
   x1 := Round(AxGlb+BxGlb*x);
   y1 := Round(AyGlb+ByGlb*y);
  end;

procedure Line(x,y: real);
 var DeltaX,DeltaY,XStep,YStep,direction,x2,y2  : integer;
  begin
   x2 := Round(AxGlb+BxGlb*x);
   y2 := Round(AyGlb+ByGlb*y);
   if x1>x2 then XStep := -1 else XStep := 1;
   if y1>y2 then YStep := -1 else YStep := 1;
   DeltaX := abs(x2-x1);
   DeltaY := abs(y2-y1);
   if DeltaX=0 then direction := -1
               else direction := 0;
   while not ((x1=x2) and (y1=y2)) do
    begin
     if (y1>=0) and (y1<YScreenMax-3) then
      case LineStyle of
       0   : DrawPoint(x1,y1);
       1   : case dot of
               1 : DrawPoint(x1,y1);
               3 : Dot := -1;
             end;
       2,6 : case dot of
               2..12 : DrawPoint(x1,y1);
               15    : Dot := -1;
             end;
       3,7 : case dot of
               0..30,36..42 : DrawPoint(x1,y1);
               47           : Dot := -1;
             end;
       4,8 : case dot of
               0..30,36,42 : DrawPoint(x1,y1);
               47          : Dot := -1;
             end;
       5,9 : case dot of
               0..33,39,45,46 : DrawPoint(x1,y1);
               47             : begin DrawPoint(x1,y1); Dot := -1; end;
             end;
      end;
     if LineStyle<>0 then dot := dot+1;
     if direction<0 then
      begin
       y1 := y1+YStep;
       direction := direction+DeltaX;
      end
     else
      begin
       x1 := x1+XStep;
       direction := direction-DeltaY;
      end;
    end;
  end;

procedure DrawLine(x1,y1,x2,y2: real);
  begin
   pset(x1,y1);
   Line(x2,y2);
  end;

procedure Axis(Zero: boolean;ScaleX,ScaleY: integer);
 var i,j    : integer;
```

```
    Xx,Yy  : real;
  begin
   SetLineStyle(0);
   Xx := X_max-X_min;
   Yy := Y_max-Y_min;
   DrawLine(X_max,Y_min,X_min,Y_min);
   DrawLine(X_min,Y_max,X_min,Y_min);
   if Zero then
    begin
     if (Y_min<0) and (Y_max>=0) then
      for j := 0 to 60 do
       begin
        pset(X_min+Xx*j/60,0);
        DrawPoint(x1,y1);
       end;
     if (X_min<0) and (X_max>=0) then
      for j := 0 to 40 do
       begin
        pset(0,Y_min+Yy*j/40);
        DrawPoint(x1,y1);
       end;
    end;
   for j := 0 to ScaleX do
    begin
     Sx[j] := X_min+Xx*j/ScaleX;
     pset(Sx[j],Y_min);
     for i := 1 to 3 do DrawPoint(x1,y1+i);
    end;
   for j := 0 to ScaleY do
    begin
     Sy[j] := Y_min+Yy*j/ScaleY;
     pset(X_min,Sy[j]);
     for i := 1 to 3 do DrawPoint(x1-i,y1);
    end;
   SwapScreen;
   gotoxy(1,1);
   Write('ScaleX  :');
   for i := 0 to ScaleX do
    begin
     if (i mod 8)=0 then writeln;
     if logs then Write(power(10,Sx[i]):7:1,' ,')
             else Write(Sx[i]:7:1,' ,');
    end;
   writeln; writeln;
   Write('ScaleY  :');
   for i := 0 to ScaleY do
    begin
     if (i mod 8)=0 then writeln;
     Write(Sy[i]:7:1,' ,');
    end;
   writeln; writeln;
   Delay(1000);
   SwapScreen;
  end;

procedure plot_function;
 var i,j,n,fn,k               : integer;
     delta,w,a,b,zz,max,min,T,F  : real;
     err1,err2,logs              : boolean;
 label err;
  begin
   GraphicsMode;
   SelectScreen(2);
   ClearScreen;
   SelectScreen(1);
```

```
  logs := false;
  fn := 3;
  n := 100;
  case num of
   '8' : begin
          DefineWorld(0, -0.2, 2, 1.2);
          Axis(true,3,13);
         end;
   '7' : begin
          DefineWorld(0,-0.5,4.7,4.5);
          Axis(true,3,9);
         end;
   '1' : begin
          DefineWorld(0,-0.25,1.5,0.25);
          Axis(true,3,10);
         end;

  end;
  delta := (X_max-X_min)/n;
  for j := 1 to fn do
   begin
    max := -1e20;
    min := 1e20;
    err1 := false;
    err2 := false;
    SetLineStyle(j);
    for i := 0 to n do
     begin
      a := X_min+delta*i;
      if logs then w := power(10,a) else w := a;
      case num of

       '8' : case j of
           1 : begin
           if w <= 1 then
  b := (sin(w*pi))*(sin(w*pi))
               ELSE  B := 0;
             end;
           2 : begin
         if w <= 100 then
              b := (
{TUT m = 0.84}
              2*(0.1565)*exp(-2.2302*w)*cos(pi*w*0.7961)
              + 2*(1.1339)*exp(-2.2302*w)*sin(pi*w*0.7961)
              + 2*(-0.1596)*exp(-1.5701*w)*cos(pi*w*2.2760)
              + 2*(-0.3463)*exp(-1.5701*w)*sin(pi*w*2.2760))
              ELSE  B := 0;
              end;

           3 : begin
    if (w<=100)or(b>0) then
              b :=(
{TUT Plot Optimun}
2*(A1)*(exp(pole1*w))*(cos(pole2*w*pi))+2*(B1)*(exp(pole1*w))*(sin(pole2*w*pi))
+2*(A2)*(exp(pole3*w))*(cos(pole4*w*pi))+2*(B2)*(exp(pole3*w))*(sin(pole4*w*pi)))

              ELSE  B := 0;
              end;

           4 : begin
               end;
          end;
  end;
  if (b<-1e20) then
        begin
         err1 := true;
```

```
          goto err;
         end;
        if (b>1e20) then
         begin
          err2 := true;
          goto err;
         end;
        if logs then b := 10*log(b);
        if min>b then min := b;
        if max<b then max := b;
        if i=0 then pset(a,b) else Line(a,b);
       end;
       err:
       SwapScreen;
       Write('function ',j,'  :    Min ');
       if err1 then write('<') else write('=');
       Write(min:8:2,'   Max ');
       if err2 then write('>') else write('=');
       Writeln(max:8:2);
       Delay(1000);
       if j<>fn then SwapScreen;
      end;
     Sound(2000);
     Delay(70);
     NoSound;
     SwapScreen;
     wait(1);
     TextMode;
    end;

 begin
  TextMode;
  AllocateRAMScreen;
  ColorGlb := 255;
   ClrScr;
num := '8';
plot_function;
 end;


procedure Plot_2D_Function_Error;

 function log(x:real):real;
   begin
    log := ln(x) / ln(10);
   end;

 function fact(n:integer):real;
  var x  : real;
   begin
    x := 1;
    for n := n downto 1 do
     x := x*n;
    fact := x;
   end;

 function power(x,n:real):real;    { function power(base,order) }
   begin
    if x=0 then power := 0
    else
     if x>0 then  power := exp(ln(x)*n)
     else
      if frac(n)=0 then
       begin
        if odd(trunc(n)) then power := -exp(ln(abs(x))*n)
        else power := exp(ln(abs(x))*n);
```

```
      end
     else
      begin
       writeln('*** error ***');
       read;
      end;
  end;

var Sx,Sy                      : array[0..30] of real;
    X_min,X_max,Y_min,Y_max,
    AxGlb,AyGlb,BxGlb,ByGlb    : real;
    x1,y1                      : integer;
    LineStyle,dot              : byte;
    num                        : char;
    logs                       : boolean;

procedure DefineWorld(X1W,Y2W,X2W,Y1W: real);
 var X_1,Y_1,X_2,Y_2  : integer;
  begin
   X_min := X1W;
   Y_min := Y2W;
   X_max := X2W;
   Y_max := Y1W;
   X_1 := 3;
   Y_1 := 0;
   X_2 := XScreenMax-1;
   Y_2 := YScreenMax-4;
   BxGlb := (X_2-X_1)/(X2W-X1W);
   ByGlb := (Y_2-Y_1)/(Y2W-Y1W);
   AxGlb := X_1-X1W*BxGlb;
   AyGlb := Y_1-Y1W*ByGlb;
  end;

procedure SetLineStyle(m: byte);
  begin
   LineStyle := m;
   dot := 0;
  end;

procedure pset(x,y: real);
  begin
   x1 := Round(AxGlb+BxGlb*x);
   y1 := Round(AyGlb+ByGlb*y);
  end;

procedure Line(x,y: real);
 var DeltaX,DeltaY,XStep,YStep,direction,x2,y2  : integer;
  begin
   x2 := Round(AxGlb+BxGlb*x);
   y2 := Round(AyGlb+ByGlb*y);
   if x1>x2 then XStep := -1 else XStep := 1;
   if y1>y2 then YStep := -1 else YStep := 1;
   DeltaX := abs(x2-x1);
   DeltaY := abs(y2-y1);
   if DeltaX=0 then direction := -1
               else direction := 0;
   while not ((x1=x2) and (y1=y2)) do
    begin
     if (y1>=0) and (y1<YScreenMax-3) then
      case LineStyle of
       0   : DrawPoint(x1,y1);
       1   : case dot of
               1 : DrawPoint(x1,y1);
               3 : Dot := -1;
             end;
```

```
      2,6 : case dot of
             2..12 : DrawPoint(x1,y1);
             15    : Dot := -1;
            end;
      3,7 : case dot of
             0..30,36..42 : DrawPoint(x1,y1);
             47           : Dot := -1;
            end;
      4,8 : case dot of
             0..30,36,42 : DrawPoint(x1,y1);
             47          : Dot := -1;
            end;
      5,9 : case dot of
             0..33,39,45,46 : DrawPoint(x1,y1);
             47             : begin DrawPoint(x1,y1); Dot := -1; end;
            end;
     end;
    if LineStyle<>0 then dot := dot+1;
    if direction<0 then
     begin
      y1 := y1+YStep;
      direction := direction+DeltaX;
     end
    else
     begin
      x1 := x1+XStep;
      direction := direction-DeltaY;
     end;
   end;
 end;

procedure DrawLine(x1,y1,x2,y2: real);
 begin
  pset(x1,y1);
  Line(x2,y2);
 end;

procedure Axis(Zero: boolean;ScaleX,ScaleY: integer);
 var i,j    : integer;
     Xx,Yy  : real;
 begin
  SetLineStyle(0);
  Xx := X_max-X_min;
  Yy := Y_max-Y_min;
  DrawLine(X_max,Y_min,X_min,Y_min);
  DrawLine(X_min,Y_max,X_min,Y_min);
  if Zero then
   begin
    if (Y_min<0) and (Y_max>=0) then
     for j := 0 to 60 do
      begin
       pset(X_min+Xx*j/60,0);
       DrawPoint(x1,y1);
      end;
    if (X_min<0) and (X_max>=0) then
     for j := 0 to 40 do
      begin
       pset(0,Y_min+Yy*j/40);
       DrawPoint(x1,y1);
      end;
   end;
  for j := 0 to ScaleX do
   begin
    Sx[j] := X_min+Xx*j/ScaleX;
    pset(Sx[j],Y_min);
```

```
      end
     else
      begin
       writeln('*** error ***');
       read;
      end;
  end;

var Sx,Sy                      : array[0..30] of real;
    X_min,X_max,Y_min,Y_max,
    AxGlb,AyGlb,BxGlb,ByGlb    : real;
    x1,y1                      : integer;
    LineStyle,dot              : byte;
    num                        : char;
    logs                       : boolean;

procedure DefineWorld(X1W,Y2W,X2W,Y1W: real);
 var X_1,Y_1,X_2,Y_2  : integer;
  begin
   X_min := X1W;
   Y_min := Y2W;
   X_max := X2W;
   Y_max := Y1W;
   X_1 := 3;
   Y_1 := 0;
   X_2 := XScreenMax-1;
   Y_2 := YScreenMax-4;
   BxGlb := (X_2-X_1)/(X2W-X1W);
   ByGlb := (Y_2-Y_1)/(Y2W-Y1W);
   AxGlb := X_1-X1W*BxGlb;
   AyGlb := Y_1-Y1W*ByGlb;
  end;

procedure SetLineStyle(m: byte);
  begin
   LineStyle := m;
   dot := 0;
  end;

procedure pset(x,y: real);
  begin
   x1 := Round(AxGlb+BxGlb*x);
   y1 := Round(AyGlb+ByGlb*y);
  end;

procedure Line(x,y: real);
 var DeltaX,DeltaY,XStep,YStep,direction,x2,y2  : integer;
  begin
   x2 := Round(AxGlb+BxGlb*x);
   y2 := Round(AyGlb+ByGlb*y);
   if x1>x2 then XStep := -1 else XStep := 1;
   if y1>y2 then YStep := -1 else YStep := 1;
   DeltaX := abs(x2-x1);
   DeltaY := abs(y2-y1);
   if DeltaX=0 then direction := -1
               else direction := 0;
   while not ((x1=x2) and (y1=y2)) do
    begin
     if (y1>=0) and (y1<YScreenMax-3) then
      case LineStyle of
       0   : DrawPoint(x1,y1);
       1   : case dot of
               1 : DrawPoint(x1,y1);
               3 : Dot := -1;
             end;
```

```
    for i := 1 to 3 do DrawPoint(x1,y1+i);
   end;
  for j := 0 to ScaleY do
   begin
    Sy[j] := Y_min+Yy*j/ScaleY;
    pset(X_min,Sy[j]);
    for i := 1 to 3 do DrawPoint(x1-i,y1);
   end;
  SwapScreen;
  gotoxy(1,1);
  Write('ScaleX  :');
  for i := 0 to ScaleX do
   begin
    if (i mod 8)=0 then writeln;
    if logs then Write(power(10,Sx[i]):7:1,' ,')
            else Write(Sx[i]:7:1,' ,');
   end;
  writeln; writeln;
  Write('ScaleY  :');
  for i := 0 to ScaleY do
   begin
    if (i mod 8)=0 then writeln;
    Write(Sy[i]:7:1,' ,');
   end;
  writeln; writeln;
  Delay(1000);
  SwapScreen;
 end;

procedure plot_function;
 var i,j,n,fn,k                : integer;
     delta,w,a,b,zz,max,min,T,F  : real;
     err1,err2,logs            : boolean;
 label err;
  begin
   GraphicsMode;
   SelectScreen(2);
   ClearScreen;
   SelectScreen(1);
   logs := false;
   fn := 3;
   n := 100;
   case num of
    '8' : begin
           DefineWorld(0, -0.1, 1, 0.1);
           Axis(true,5,10);
          end;
    '7' : begin
           DefineWorld(0,-0.5,4.7,4.5);
           Axis(true,3,9);
          end;
    '1' : begin
           DefineWorld(0,-0.25,1.5,0.25);
           Axis(true,3,10);
          end;

   end;
   delta := (X_max-X_min)/n;
   for j := 1 to fn do
    begin
     max := -1e20;
     min := 1e20;
     err1 := false;
     err2 := false;
     SetLineStyle(j);
     for i := 0 to n do
```

```
       begin
        a := X_min+delta*i;
        if logs then w := power(10,a) else w := a;
        case num of

         '8' : case j of
             1 : begin
                 end;
             2 : begin
           if w <= 100 then
                  b := (sin(w*pi))*(sin(w*pi)) - (
{TUT m = 0.84}
                 2*(0.1565)*exp(-2.2302*w)*cos(pi*w*0.7961)
                 + 2*(1.1339)*exp(-2.2302*w)*sin(pi*w*0.7961)
                 + 2*(-0.1596)*exp(-1.5701*w)*cos(pi*w*2.2760)
                 + 2*(-0.3463)*exp(-1.5701*w)*sin(pi*w*2.2760))
                 ELSE  B := 0;
                 end;
             3 : begin
     if (w<=100)or(b>0) then
                 b := (sin(w*pi))*(sin(w*pi)) - (
{TUT Plot Optimun}
2*(A1)*(exp(pole1*w))*(cos(pole2*w*pi))+2*(B1)*(exp(pole1*w))*(sin(pole2*w*pi))
+2*(A2)*(exp(pole3*w))*(cos(pole4*w*pi))+2*(B2)*(exp(pole3*w))*(sin(pole4*w*pi)))
                 ELSE  B := 0;
                 end;
             4 : begin
                 end;
               end;
 end;
if (b<-1e20) then
         begin
          err1 := true;
          goto err;
         end;
        if (b>1e20) then
         begin
          err2 := true;
          goto err;
         end;
        if logs then b := 10*log(b);
        if min>b then min := b;
        if max<b then max := b;
        if i=0 then pset(a,b) else Line(a,b);
       end;
       err:
       SwapScreen;
       Write('function ',j,'  :    Min ');
       if err1 then write('<') else write('=');
       Write(min:8:2,'   Max ');
       if err2 then write('>') else write('=');
       Writeln(max:8:2);
       Delay(1000);
       if j<>fn then SwapScreen;
     end;
    Sound(2000);
    Delay(70);
    NoSound;
    SwapScreen;
    wait(1);
    TextMode;
   end;

 begin
  TextMode;
```

```
  AllocateRAMScreen;
  ColorGlb := 255;
   ClrScr;
num := '8';
plot_function;
 end;
```

```
{-------- This is the function to integrate --------}
function TargetF1(X : real) : real;
begin
TargetF1 := -2*(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi))
             *(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi));
end;
function TargetF2(X : real) : real;
begin
TargetF2 := -2*(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi))
             *(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi));
end;
function TargetF3(X : real) : real;
begin
TargetF3 := -2*(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi))
             *(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi));
end;
function TargetF4(X : real) : real;
begin
TargetF4 := -2*(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi))
             *(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi));
end;

function TargetF5(X : real) : real;
begin
TargetF5 := -2*(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi))
             *(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi));
end;
function TargetF6(X : real) : real;
begin
TargetF6 := -2*(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi))
             *(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi));
end;
function TargetF7(X : real) : real;
begin
TargetF7 := -2*(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi))
             *(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi));
end;
function TargetF8(X : real) : real;
begin
TargetF8 := -2*(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi))
             *(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi));
end;
function TargetF9(X : real) : real;
begin
TargetF9 := -2*(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi))
             *(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi));
end;
function TargetF10(X : real) : real;
begin
TargetF10 := -2*(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi))
              *(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi));
end;
function TargetF11(X : real) : real;
begin
TargetF11 := -2*(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi))
              *(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi));
end;
function TargetF12(X : real) : real;
begin
TargetF12 := -2*(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi))
              *(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi));
end;
function TargetF13(X : real) : real;.
```

```
begin
TargetF13 := -2*(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi))
             *(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi));
end;
function TargetF14(X : real) : real;
begin
TargetF14 := -2*(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi))
             *(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi));
end;
function TargetF15(X : real) : real;
begin
TargetF15 := -2*(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi))
             *(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi));
end;
function TargetF16(X : real) : real;
begin
TargetF16 := -2*(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi))
             *(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi));
end;

function TargetF17(X : real) : real;
begin
TargetF17 := -(sin(x*pi))*(sin(x*pi))*(exp(pole[1,1]*x))*(cos(pole[1,2]*x*pi));
end;
function TargetF18(X : real) : real;
begin
TargetF18 := -(sin(x*pi))*(sin(x*pi))*(exp(pole[1,1]*x))*(sin(pole[1,2]*x*pi));
end;
function TargetF19(X : real) : real;
begin
TargetF19 := -(sin(x*pi))*(sin(x*pi))*(exp(pole[2,1]*x))*(cos(pole[2,2]*x*pi));
end;
function TargetF20(X : real) : real;
begin
TargetF20 := -(sin(x*pi))*(sin(x*pi))*(exp(pole[2,1]*x))*(sin(pole[2,2]*x*pi));
end;
{-------------------------------------------------}
procedure Resultg(Dimen        : integer;
                  Coefficients : matrix;
                  Constants    : vector;
                  Solution     : vector;
                  Error        : byte);
var Column,Row : integer;
begin
  case Error of
    0 : begin
          Writeln('The solution:');
          for Row := 1 to Dimen do
            Writeln('A',row,' => ',Solution[Row]);
          Writeln;
        end;
    1 : Writeln('The dimension of the matrix must be greater than 1.');
    2 : Writeln('There is no solution to this set of equations.');
  end;
A1:=Solution[1];
B1:=Solution[2];
A2:=Solution[3];
B2:=Solution[4];
end; { procedure Result }


procedure Simpson1(LowerLimit   : real;
                   UpperLimit   : real;
                   NumIntervals : integer;
                   var Integral : real);
```

```
              var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF1(Point) + 4 * TargetF1(Point + Spacing)
              + TargetF1(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }

procedure Simpson2(LowerLimit   : real;
                   UpperLimit   : real;
                   NumIntervals : integer;
                 var Integral   : real;
                 var Error      : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF2(Point) + 4 * TargetF2(Point + Spacing)
              + TargetF2(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson3(LowerLimit   : real;
                   UpperLimit   : real;
                   NumIntervals : integer;
                 var Integral   : real;
                 var Error      : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
```

```
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF3(Point) + 4 * TargetF3(Point + Spacing)
               + TargetF3(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson4(LowerLimit   : real;
                   UpperLimit   : real;
                   NumIntervals : integer;
                var Integral    : real;
                var Error       : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF4(Point) + 4 * TargetF4(Point + Spacing)
               + TargetF4(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson5(LowerLimit   : real;
                   UpperLimit   : real;
                   NumIntervals : integer;
                var Integral    : real;
                var Error       : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF5(Point) + 4 * TargetF5(Point + Spacing)
               + TargetF5(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }

procedure Simpson6(LowerLimit   : real;
                   UpperLimit   : real;
                   NumIntervals : integer;
```

```
                var Integral     : real;
                var Error        : byte);
  var Spacing,Point,Sum  : real;
      Interval           : integer;
  begin
    if NumIntervals <= 0 then Error := 1
    else
      if odd(NumIntervals) then Error := 2
      else Error := 0;
    if Error = 0 then
      begin
        Spacing := (UpperLimit - LowerLimit) / NumIntervals;
        Point := LowerLimit;
        Sum := 0;
        for Interval := 0 to NumIntervals div 2 - 1 do
        begin
          Sum := Sum + TargetF6(Point) + 4 * TargetF6(Point + Spacing)
                 + TargetF6(Point + 2 * Spacing);
          Point := Point + 2 * Spacing;
        end;
        Integral := Spacing * Sum / 3;
      end;
  end; { procedure Simpson }
  procedure Simpson7(LowerLimit   : real;
                     UpperLimit   : real;
                     NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
  var Spacing,Point,Sum  : real;
      Interval           : integer;
  begin
    if NumIntervals <= 0 then Error := 1
    else
      if odd(NumIntervals) then Error := 2
      else Error := 0;
    if Error = 0 then
      begin
        Spacing := (UpperLimit - LowerLimit) / NumIntervals;
        Point := LowerLimit;
        Sum := 0;
        for Interval := 0 to NumIntervals div 2 - 1 do
        begin
          Sum := Sum + TargetF7(Point) + 4 * TargetF7(Point + Spacing)
                 + TargetF7(Point + 2 * Spacing);
          Point := Point + 2 * Spacing;
        end;
        Integral := Spacing * Sum / 3;
      end;
  end; { procedure Simpson }
  procedure Simpson8(LowerLimit   : real;
                     UpperLimit   : real;
                     NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
  var Spacing,Point,Sum  : real;
      Interval           : integer;
  begin
    if NumIntervals <= 0 then Error := 1
    else
      if odd(NumIntervals) then Error := 2
      else Error := 0;
    if Error = 0 then
      begin
        Spacing := (UpperLimit - LowerLimit) / NumIntervals;
        Point := LowerLimit;
```

```
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF8(Point) + 4 * TargetF8(Point + Spacing)
               + TargetF8(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }

procedure Simpson9(LowerLimit   : real;
                   UpperLimit   : real;
                   NumIntervals : integer;
              var Integral      : real;
              var Error         : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF9(Point) + 4 * TargetF9(Point + Spacing)
               + TargetF9(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }

procedure Simpson10(LowerLimit   : real;
                   UpperLimit   : real;
                   NumIntervals : integer;
              var Integral      : real;
              var Error         : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF10(Point) + 4 * TargetF10(Point + Spacing)
               + TargetF10(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson11(LowerLimit   : real;
```

```
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
 var Spacing,Point,Sum  : real;
     Interval           : integer;
 begin
   if NumIntervals <= 0 then Error := 1
   else
     if odd(NumIntervals) then Error := 2
     else Error := 0;
   if Error = 0 then
     begin
       Spacing := (UpperLimit - LowerLimit) / NumIntervals;
       Point := LowerLimit;
       Sum := 0;
       for Interval := 0 to NumIntervals div 2 - 1 do
       begin
         Sum := Sum + TargetF11(Point) + 4 * TargetF11(Point + Spacing)
                + TargetF11(Point + 2 * Spacing);
         Point := Point + 2 * Spacing;
       end;
       Integral := Spacing * Sum / 3;
     end;
 end; { procedure Simpson }
 procedure Simpson12(LowerLimit   : real;
                     UpperLimit   : real;
                     NumIntervals : integer;
                 var Integral     : real;
                 var Error        : byte);
 var Spacing,Point,Sum  : real;
     Interval           : integer;
 begin
   if NumIntervals <= 0 then Error := 1
   else
     if odd(NumIntervals) then Error := 2
     else Error := 0;
   if Error = 0 then
     begin
       Spacing := (UpperLimit - LowerLimit) / NumIntervals;
       Point := LowerLimit;
       Sum := 0;
       for Interval := 0 to NumIntervals div 2 - 1 do
       begin
         Sum := Sum + TargetF12(Point) + 4 * TargetF12(Point + Spacing)
                + TargetF12(Point + 2 * Spacing);
         Point := Point + 2 * Spacing;
       end;
       Integral := Spacing * Sum / 3;
     end;
 end; { procedure Simpson }
 procedure Simpson13(LowerLimit   : real;
                     UpperLimit   : real;
                     NumIntervals : integer;
                 var Integral     : real;
                 var Error        : byte);
 var Spacing,Point,Sum  : real;
     Interval           : integer;
 begin
   if NumIntervals <= 0 then Error := 1
   else
     if odd(NumIntervals) then Error := 2
     else Error := 0;
   if Error = 0 then
     begin
```

```
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF13(Point) + 4 * TargetF13(Point + Spacing)
               + TargetF13(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }

procedure Simpson14(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF14(Point) + 4 * TargetF14(Point + Spacing)
               + TargetF14(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson15(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF15(Point) + 4 * TargetF15(Point + Spacing)
               + TargetF15(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
```

```
procedure Simpson16(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF16(Point) + 4 * TargetF16(Point + Spacing)
               + TargetF16(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson17(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF17(Point) + 4 * TargetF17(Point + Spacing)
               + TargetF17(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson18(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
```

```
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF18(Point) + 4 * TargetF18(Point + Spacing)
               + TargetF18(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson19(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF19(Point) + 4 * TargetF19(Point + Spacing)
               + TargetF19(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpson20(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + TargetF20(Point) + 4 * TargetF20(Point + Spacing)
               + TargetF20(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
```

```
procedure Gaussian_Elimination(Dimen        : integer;
                               Coefficients : matrix;
                               Constants    : vector;
                           var Solution     : vector;
                           var Error        : byte);
const NearlyZero = 1E-10;
var Term,Row,ReferenceRow  : integer;
    Sum,Multiplier         : real;

procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            var Coefficients : matrix;
            var Constants    : vector;
            var Error        : byte);
var NewRow    : integer;
    Dummy     : real;
    DummyRow  : vector;
begin
  Error := 2;
  NewRow := ReferenceRow;
  while (Error > 0) and (NewRow < Dimen) do
  begin
    NewRow := Succ(NewRow);
    if ABS(Coefficients[NewRow,ReferenceRow]) > NearlyZero then
    begin
      DummyRow := Coefficients[NewRow];
      Coefficients[NewRow] := Coefficients[ReferenceRow];
      Coefficients[ReferenceRow] := DummyRow;
      Dummy := Constants[NewRow];
      Constants[NewRow] := Constants[ReferenceRow];
      Constants[ReferenceRow] := Dummy;
      Error := 0;
    end;
  end;
end; { procedure Pivot }

begin { procedure Gaussian_Elimination }
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    if Dimen = 1 then
      if ABS(Coefficients[1,1]) < NearlyZero then
        Error := 2
      else
        Solution[1] := Constants[1] / Coefficients[1,1];
  if Dimen > 1 then
    begin
      ReferenceRow := 0;
      while (Error = 0) and (ReferenceRow < Dimen-1) do
      begin
        ReferenceRow := Succ(ReferenceRow);
        if ABS(Coefficients[ReferenceRow,ReferenceRow]) < NearlyZero then
          Pivot(Dimen,ReferenceRow,Coefficients,Constants,Error);
        if Error = 0 then
          for Row := ReferenceRow + 1 to Dimen do
            if ABS(Coefficients[Row,ReferenceRow]) > NearlyZero then
            begin
              Multiplier := -Coefficients[Row,ReferenceRow] /
                            Coefficients[ReferenceRow,ReferenceRow];
              for Term := 1 to Dimen do
                Coefficients[Row,Term] := Coefficients[Row,Term]
                                        + Multiplier * Coefficients[ReferenceRow,Term];
              Constants[Row] := Constants[Row]
```

```
                                + Multiplier * Constants[ReferenceRow];
            end;
        end;
        if ABS(Coefficients[Dimen,Dimen]) < NearlyZero then
          Error := 2;
        if Error = 0 then
          begin
            Term := Dimen;
            while Term >= 1 do
            begin
              Sum := 0;
              for Row := Term + 1 to Dimen do
                Sum := Sum + Coefficients[Term,Row] * Solution[Row];
              Solution[Term] := (Constants[Term]-Sum) / Coefficients[Term,Term];
              Term := Pred(Term);
            end;
      end;
    end;
  end; { procedure Gaussian_Elimination }
```

```
{-------- This is the Error function to integrate --------}
function ErrorTarget(X : real) : real;
begin
ErrorTarget := sin(x*pi)*sin(x*pi)-2*A1*exp(pole1*x)*cos(pole2*x*pi)
-2*B1*exp(pole1*x)*sin(pole2*x*pi)-2*A2*exp(pole3*x)*cos(pole4*x*pi)
-2*B2*exp(pole3*x)*sin(pole4*x*pi);
end;
procedure SimpsonErr(LowerLimit   : real;
                     UpperLimit   : real;
                     NumIntervals : integer;
                 var Integral     : real;
                 var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + ErrorTarget(Point) + 4 * ErrorTarget(Point + Spacing)
             + ErrorTarget(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }


{-------- This is the function to integrate --------}

function dE_dpole1(X : real) : real;
begin
dE_dpole1 :=
  (2*A1*x*exp(-pole1*x)*cos(pole2*x*pi)+2*B1*x*exp(-pole1*x)*sin(pole2*x*pi));
end;
function dE_dpole2(X : real) : real;
begin
dE_dpole2 :=
(2*A1*x*pi*exp(-pole1*x)*sin(pole2*x*pi)-2*B1*x*pi*exp(pole1*x)*cos(pole2*x*pi));
end;
function dE_dpole3(X : real) : real;
begin
dE_dpole3 :=
(2*A2*x*exp(-pole3*x)*cos(pole4*x*pi)+2*B2*x*exp(-pole3*x)*sin(pole4*x*pi));
end;
function dE_dpole4(X : real) : real;
begin
dE_dpole4 :=
(2*A2*x*pi*exp(-pole3*x)*sin(pole4*x*pi)-2*B2*x*pi*exp(pole3*x)*cos(pole4*x*pi));
end;

{------------------------ dominator-------------------------}
procedure GotData(var Dimen             : integer;
                  var Coefficients,Pole : matrix;
                  var Constants         : vector);
                  var Row,Col           : integer;
begin
  Writeln;
```

```
  repeat
    Write('Number of Poles : ');
{    Write('Dimension of the coefficient matrix (1-',ArraySize,')? ');}
    Readln(Dimen);
    Realpoles := Round(Dimen/2)
  until (Dimen >= 1) and (Dimen <= ArraySize);
  Writeln;
  for Row := 1 to Realpoles do
  for col := 1 to Dimen do
      begin
        Write('Pole [',Row,',',Col,']: ');
        Readln(Pole[Row,Col]);
      end;


end; { procedure GotData }

procedure Result(Dimen        : integer;
                 Coefficients : matrix;
                 Constants    : vector;
                 Solution     : vector;
                 Error        : byte);
var Column,Row : integer;
begin
  Writeln;
  Writeln('The coefficients: ');
  for Row := 1 to Dimen do
  begin
    for Column := 1 to Dimen do
      Write( Coefficients[Row,Column]:14:8);
    Writeln;
  end;
  Writeln;
  Writeln('The constants:');
  for Row := 1 to Dimen do
    Writeln(Constants[Row]);
  Writeln;
  case Error of
    0 : begin
          Writeln('The solution:');
          for Row := 1 to Dimen do
            Writeln('A',row,' => ',Solution[Row]);
          Writeln;
        end;
    1 : Writeln('The dimension of the matrix must be greater than 1.');
    2 : Writeln('There is no solution to this set of equations.');
  end;
end; { procedure Result }

procedure Simpsonp5(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
```

```
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + dE_dpole1(Point) + 4 * dE_dpole1(Point + Spacing)
               + dE_dpole1(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }

procedure Simpsonp6(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + dE_dpole2(Point) + 4 * dE_dpole2(Point + Spacing)
               + dE_dpole2(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpsonp7(LowerLimit   : real;
                    UpperLimit   : real;
                    NumIntervals : integer;
                var Integral     : real;
                var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + dE_dpole3(Point) + 4 * dE_dpole3(Point + Spacing)
               + dE_dpole3(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }
procedure Simpsonp8(LowerLimit   : real;
                    UpperLimit   : real;
```

```
                NumIntervals : integer;
              var Integral     : real;
              var Error        : byte);
var Spacing,Point,Sum  : real;
    Interval           : integer;
begin
  if NumIntervals <= 0 then Error := 1
  else
    if odd(NumIntervals) then Error := 2
    else Error := 0;
  if Error = 0 then
    begin
      Spacing := (UpperLimit - LowerLimit) / NumIntervals;
      Point := LowerLimit;
      Sum := 0;
      for Interval := 0 to NumIntervals div 2 - 1 do
      begin
        Sum := Sum + dE_dpole4(Point) + 4 * dE_dpole4(Point + Spacing)
               + dE_dpole4(Point + 2 * Spacing);
        Point := Point + 2 * Spacing;
      end;
      Integral := Spacing * Sum / 3;
    end;
end; { procedure Simpson }


procedure GetData(var LowerLimit   : real;
                  var UpperLimit   : real;
                  var NumIntervals : integer);
begin
      lowerlimit := 0;
   {   upperlimit := 3.1416;}
      upperlimit := 1;
      numintervals := 10;
 { Write('Lower limit of integration? ');}
 { Readln(LowerLimit);}
 { Write('Upper limit of integration? ');}
 { Readln(UpperLimit);}
 { Write('Number of intervals ? ');}
 { Readln(NumIntervals);}
end; { procedure GetData }

procedure Results(LowerLimit   : real;
                  UpperLimit   : real;
                  NumIntervals : integer;
                  Integral     : real;
                  Error        : byte);
begin
  Writeln;
{  Writeln('Lower Limit:':25,LowerLimit:25);}
{  Writeln('Upper Limit:':25,UpperLimit:25);}
{  Writeln('Number of intervals:':25,NumIntervals:5);}
{  Writeln;}
  case Error of
    0 : Writeln('Integral:':25,Integral:25);
    1 : Writeln('The number of intervals must be greater than 0.');
    2 : Writeln('The number of intervals must be even number');
  end;
end; { procedure Results }
```

# ประวัติผู้เขียน

| | |
|---|---|
| ชื่อผู้เขียน | นาย ธนิตพงษ์ วิบูลยานนท์ |
| วันเดือนปีเกิด | วันที่ 25 พฤษภาคม 2506 |
| วุฒิการศึกษาระดับปริญญาตรี | อุสาหกรรมศาสตร์บัณฑิต สาขาเทคโนโลยี่อีเลคทรอนิคส์ |
| สถานที่สำเร็จการศึกษา | ปีการศึกษา 2531 |
| ผลงานทางวิชาการที่ได้รับการตีพิมพ์ | การประมาณฟังก์ชั่นซายน์กำลังสองโดยวิธีเน็คกาทีฟ-เกรเดียนท์ด้วยการกำหนดโพลเริ่มต้น การประชุมทางวิชาการวิศวกรรมไฟฟ้า 9 สถาบัน ครั้งที่ 14 ม.สงขลานครินทร์ |
| ประสพการณ์การทำงาน | 2529 สำนักวิจัยและบริการคอมพิวเตอร์ ส.จ.ล.<br>2531 กองช่างระบบเรดาร์ บ.วิทยุการบินฯ |
| อาชีพปัจจุบัน | วิศวกร ประจำกองช่างระบบเรดาร์ บ.วิทยุการบินฯ |