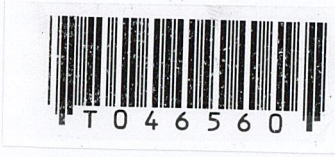
การออกแบบวงจรกรุ๊ฟดีเลย์อิควอไลเซอร์

DESIGN OF GROUP DELAY EQUALIZER



นายพิทักษ์ หมีนสันธิ
นายไพรัช จัตุรภุชพิทักษ์

.b.....
.i.....

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต
สาขาเทคโนโลยีโทรคมนาคม ภาควิชาเทคนิคอุตสาหกรรม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2544

DESIGN OF GROUP DELAY EQUALIZER

Mr. PHITHAK MUENSANTHI

Mr. PIRAT JATURAPUTPITAK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF THE TECHNOLOGY TELECOMMUNICATION

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2001

หัวข้อปริญญานิพนธ์	การออกแบบวงจรกรุ๊ฟดีเลย์อิควอไลเซอร์
	Design of Group Delay Equalizer
นักศึกษา	นายพิทักษ์ หมื่นสันธิ เลขประจำตัว 42015646
	นายไพรัช จัตุรภุชพิทักษ์ เลขประจำตัว 42015650
อาจารย์ที่ปรึกษา	รศ.ดร.กนก เจนจิระพงศ์เวช

ภาควิชา	เทคนิคอุตสาหกรรม
ปีการศึกษา	2544

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตร์บัณฑิต

คณะกรรมการสอบปริญญานิพนธ์

_____อาจารย์ที่ปรึกษา
(รศ.ดร.กนก เจนจิระพงศ์เวช)

_____กรรมการ
( )

_____กรรมการ
( )

_____กรรมการ
( )

ลิขสิทธิ์ของคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อปริญญานิพนธ์	การออกแบบวงจรกรุ๊ฟดีเลย์อิควอไลเซอร์ Design of Group Delay Equalizer for			
นักศึกษา	นายพิทักษ์	หมื่นสันธิ	เลขประจำตัว	42015646
	นายไพรัช	จัตุรภุชพิทักษ์	เลขประจำตัว	42015650
อาจารย์ที่ปรึกษา	รศ.ดร.กนก	เจนจิระพงศ์เวช		
ปีการศึกษา	2544			

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ ได้เสนอหลักการของวงจรอิควอไลเซอร์ ยกระดับหรือลดระดับผลตอบสนองทุกย่านความถี่ โดยใช้โปรแกรมเดลไฟ เวอร์ชั่น 6 ออกแบบวงจรอิควอไลเซอร์ ซึ่งนอกจากนี้แล้วยังได้เสนอหลักการในการประมาณฟังค์ชั่นของวงจรดีเลย์อิควอไลเซอร์ ซึ่งใช้หลักการของ นิวตัน-ราฟสันที่สามารถกำหนดคุณลักษณะผลตอบสนองของกรุ๊ฟดีเลย์ตามต้องการได้ ซึ่งสามารถใช้ในการชดเชยเฟสของสัญญาณ หรือแก้ไขความผิดเพี้ยนทางเวลาเพื่อให้ได้กรุ๊ฟดีเลย์คงที่ โดยการประมาณค่าเริ่มของโพลและซีโร่แบบขนาน และใช้คอมพิวเตอร์ช่วยในการคำนวณหาคำตอบของทรานเฟอร์ฟังค์ชั่นที่ถูกประมาณขึ้น เพื่อให้ได้ตำแหน่งของโพลและซีโร่ที่ดีที่สุด ในการออกแบบวงจรอิควอไลเซอร์ดังกล่าวนั้นใช้โปรแกรม เดลไฟเวอร์ชั่น 6 เป็นตัวช่วยในการคำนวณทางด้านคอมพิวเตอร์ เพื่อให้ได้การประมาณฟังค์ชั่น ของวงจรดีเลย์อิควอไลเซอร์ที่มีความผิดพลาดน้อยที่สุด

Thesis Title	Design of Group Delay Equalizer	
Student	Mr. Phithak Muensanthi	ID 42015646
	Mr. Pirat Jaturaphuchpitak	ID 42015650
Advisor	Assoc.Prof.Dr. Kanok Janjirapongwach	
Subject	Technical Industrial	
Year	2001	

ABSTRACT

This project has presented the design of group delay equalizer by the aid of computer simulation. In order to correct, the signal group delay equalizer is utilized.

Herein the method of Newton-Rafson for group delay is used the amplitude. Frequency response of this equalizer can be increased or decreased at every frequency.

This project also describe an iterative method of group delay equalization over a specified amplitude frequency response of the design equalizer. The least mean error of group delay is obtained by using program Delphi 6 to search the optimized poles and zeros.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดี ด้วยคำแนะนำและคำปรึกษาจาก รศ.ดร. กนก เจนจิระพงศ์เวช ซึ่งเป็นอาจารย์ที่ปรึกษา และ อ.ไพศาล สิทธิโยภาสกุล คณะผู้จัดทำรู้สึกซาบซึ้งในความอนุเคราะห์จากท่าน ที่ได้ประสิทธิ์ประสาทวิชาความรู้แก่ศิษย์ในการศึกษาระดับต่างๆ และ ขอ กราบขอบพระคุณเป็นอย่างสูง

ขอกราบขอบพระคุณ บิดา มารดา และ พี่น้อง ที่ให้โอกาสกับลูกสำหรับการศึกษาเล่าเรียนรวมทั้งให้กำลังใจต่าง ๆ ในการศึกษาตลอดมา

ขอขอบคุณ เพื่อน ๆ พี ๆ น้อง ๆ ทุกท่าน ที่ให้กำลังใจและช่วยเหลือในการทำปริญญานิพนธ์ฉบับนี้มาโดยตลอด

ขออำนาจคุณพระศรีรัตนตรัยและสิ่งศักดิ์สิทธิ์ทั้งหลาย ช่วยดลบันดาลให้ทุกท่านที่กล่าวถึง ประสบความสำเร็จ มีความสุขความเจริญในหน้าที่การงานทุกประการ

คณะผู้จัดทำ

สารบัญ

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 แนวความคิดและที่มา

โดยทั่วไป ในระบบการรับส่งสัญญาณการติดต่อสื่อสาร จะต้องส่งผ่านสัญญาณเข้าวงจรต่าง ๆ ซึ่งอาจทำให้ผลตอบสนองทางเฟสไม่เป็นเชิงเส้นและทำให้กรุ๊ฟดีเลย์ที่ได้ไม่คงที่ เช่น ระบบรับส่งของสัญญาณโทรทัศน์และพัลซ์ทรานส์มิสชั่น (Pulse Transmission) และฟิลเตอร์ต่างๆ ซึ่งมีความจำเป็นที่จะต้องชดเชยทั้งทางเฟสและทางขนาดของสัญญาณ โดยทำให้เฟสที่ได้เป็นเชิงเส้นให้มากที่สุดตามที่ต้องการในช่วงทรานส์มิสชั่นแบนด์ (Transmission Band) หรือทำให้กรุ๊ฟดีเลย์ (Group Delay) คงที่ เพื่อแก้ไขความผิดเพี้ยนทางเวลาเพื่อให้ได้กรุ๊ฟดีเลย์คงที่ ฉะนั้นจึงมีความจำเป็นที่จะต้องชดเชยเฟสของสัญญาณ โดยทำให้เฟสที่ได้เป็นเชิงเส้นและให้กรุ๊ฟดีเลย์ราบเรียบที่มากที่สุด

ซึ่งในโครงงานนี้ได้เสนอวิธีการออกแบบวงจรกรุ๊ฟดีเลย์อิควอไลเซอร์สำหรับแก้ความผิดเพี้ยนของกรุ๊ฟดีเลย์ เพื่อให้ได้คุณลักษณะของผลตอบสนองทางด้านขนาดและเฟสที่คงที่

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาการทำงานวงจรกรุ๊ฟดีเลย์อิควอไลเซอร์และการแก้ไขความผิดเพี้ยนทางเวลาเพื่อให้ได้กรุ๊ฟดีเลย์คงที่
2. เพื่อศึกษาการออกแบบวงจรกรุ๊ฟดีเลย์อิควอไลเซอร์
3. เพื่อศึกษาการวัดความผิดเพี้ยนของอัตราการขยายและกรุ๊ฟดีเลย์

1.3 เนื้อหาของปริญญานิพนธ์

ปริญญานิพนธ์นี้ เป็นการเขียนโปรแกรมขึ้นมา โดยใช้โปรแกรมเดลไฟ (delphi) เพื่อออกแบบกรุ๊ฟดีเลย์อิควอไลเซอร์ ซึ่งก็คือโปรแกรมที่ใช้ปรับค่ากรุ๊ฟดีเลย์นั่นเอง สามารถทำให้กรุ๊ฟดีเลย์ที่ทำการปรับแล้ว มีค่าผิดเพี้ยน(error) น้อยกว่าค่าคลาดเคลื่อน(tolerence) ที่กำหนดไว้

ในบทที่ 2 เป็นทฤษฎีและหลักการทั่วไป ของผลตอบสนองทางด้าน ขนาด เฟส และกรุ๊ฟดีเลย์ ความสัมพันธ์ระหว่างโพล-ซีโร่ ของฟังก์ชั่น ที่ทำให้เกิดผลตอบสนองต่างๆ

ในบทที่ 3 ได้แนะนำโปรแกรมเดลไฟ ซึ่งใช้เขียนโปรแกรมขึ้นมาทั้งหมด บทนี้ได้แสดงการใช้งานต่างๆของโปรแกรม การสร้างไฟล์ต่างๆขึ้นมา อธิบายพร้อมรูปให้เป็นที่เข้าใจ

ในบทที่ 4 เป็นการออกแบบวงจรกรุ๊ฟดีเลย์อิควอไลเซอร์ เริ่มจากการหาค่ากรุ๊ฟดีเลย์ แล้วทำการปรับโดยใช้วิธี นิวตัน-ราฟสัน พร้อมโฟว์ชาจ แสดงการทำงานของโปรแกรม

ในบทที่ 5 แสดงผลการทดลอง ที่ทำการทดสอบโปรแกรมที่เขียนขึ้นมา โดยปรับกรุ๊ฟดีเลย์ของ All-pall และ low-pass

ในบทที่ 6 ซึ่งเป็นบทสุดท้ายจะสรุปและวิจารณ์ผลเพื่อเป็นแนวทางในการพัฒนาต่อไป ภาคผนวก เป็นรายละเอียดของโปรแกรม และได้สอนขั้นตอนการใช้โปรแกรม

1.4 ประโยชน์ที่จะรับจากปริญญานิพนธ์

1. สามารถนำไปใช้ในการออกแบบวงจรอิควอไลเซอร์ที่สามารถยกระดับผลตอบสนองทางด้านขนาดและเฟสได้ตามที่ต้องการ
2. สามารถนำไปใช้เพื่อให้ได้คุณลักษณะทางขนาดหรือกรุ๊ฟดีเลย์ใกล้เคียงกับคุณลักษณะทางด้านขนาดหรือกรุ๊ฟดีเลย์ที่ต้องการตามอุดมคติได้อย่างมีประสิทธิภาพ
3. สามารถนำไปใช้ในการแก้ไขความผิดเพี้ยนของสัญญาณในการรับส่งสัญญาณ
4. สามารถสร้างวงจรเพื่อใช้งานจริง ซึ่งจะสามารถนำไปเป็นแนวทางการพัฒนาหรือประยุกต์ใช้งานได้อย่างกว้างขวางและมีประสิทธิภาพมากที่สุด

บทที่ 2
ทฤษฎีและหลักการ

2.1 ผลตอบสนองทางด้านขนาดและเฟส

ในส่วนนี้จะแสดงความสัมพันธ์ระหว่างโพล (pole) และ ซีโร่ (zero) ของฟังก์ชั่น และผลตอบสนองที่สภาวะคงที่ (steady – state) ของสัญญาณจำพวกไซนูนอยล์ หรือเรียกอีกอย่างหนึ่งว่าผลกระทบ ณ. ที่ตำแหน่งของโพล และ ซีโร่ ในฟอร์มของ H(s) บนแกน $j\omega$ ผลตอบสนองในสภาวะคงที่ของฟังก์ชันสามารถเขียนเป็นสมการได้ดังนี้

$$H(j\omega) = M(\omega)e^{j\phi(\omega)} \qquad (2.1)$$

ซึ่ง $M(\omega)$ คือ ผลตอบสนองทางด้านขนาดของฟังก์ชันซึ่งเป็นฟังก์ชันคู่ของ ω

$\phi(\omega)$ คือ ผลตอบสนองทางด้านเฟสของฟังก์ชัน ω

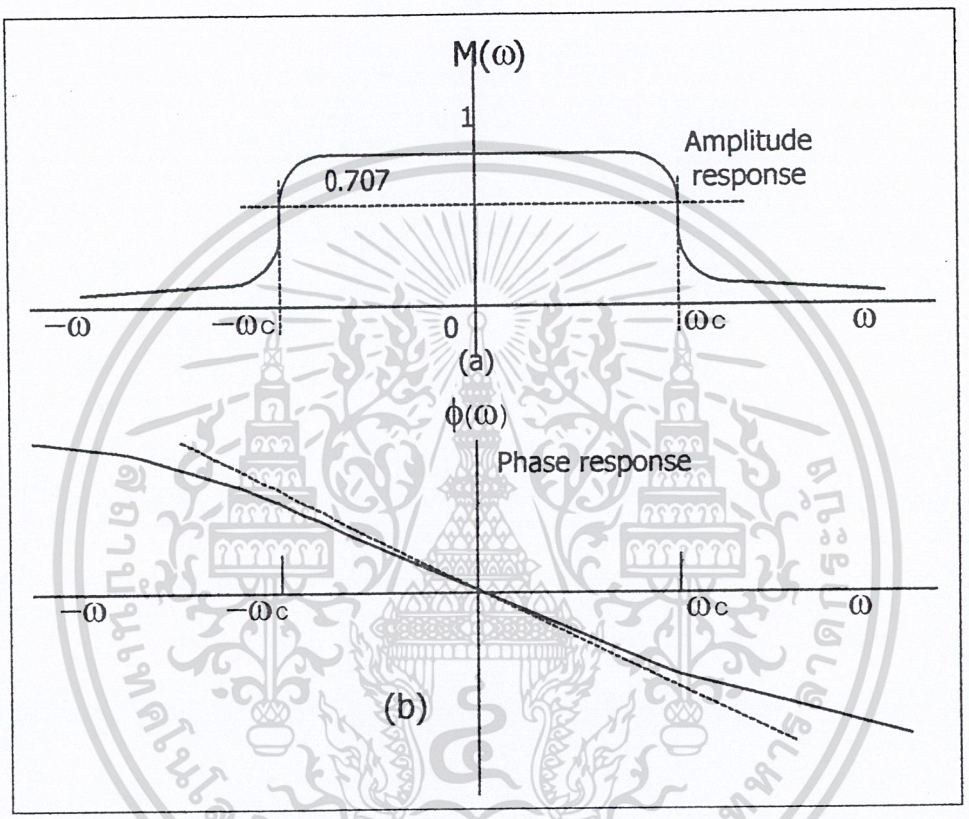
ผลตอบสนองทางด้านขนาดและเฟสของระบบจะให้ข้อมูลในการวิเคราะห์และออกแบบวงจรการส่ง พิจารณาคุณสมบัติทางด้านขนาดและเฟสของโลพาสฟิลเตอร์ (low pass filter) ซึ่งแสดงไว้ในรูปที่ 2.1a และ 2.1b ความถี่คัทออฟ (cutoff frequency) ของฟิลเตอร์บนเคอร์ฟผลตอบสนองทางด้านขนาดนั้นคือ ω_0 โดยทั่วไปแล้วความถี่ฮาร์ฟเพาเวอร์ (half power) ที่ฟังก์ชัน $|H(j\omega_{max})|$ นั้นจะมีค่าเท่ากับ 0.707 $|H(j\omega_{max})|$ และในหน่วยของเดซิเบลนั้นค่าความถี่ ณ. จุดนี้คือ 20 log $|H(j\omega_c)|$ หรือมีค่าเท่ากับ 3 เดซิเบล (db) ในรูปที่ 2.1 นั้นแสดงให้เห็นว่าที่ ณ. จุดนี้จะไม่ให้ความถี่ที่สูงกว่า ω_c ผ่านไปได้ พิจารณาขนาดของพัลซ์ (pulse) ซึ่งสเป็คตรัมแอมปลิจูดประกอบด้วยสัญญาณฮาร์มอนิก (hormonic) ที่สูงกว่า ω_c ซึ่งจะเห็นได้ว่าสัญญาณฮาร์มอนิกที่ต่ำกว่า ω_c นั้นจะสามารถผ่านไปได้แต่จะกันสัญญาณฮาร์มอนิกที่สูงกว่า ω_c จะเห็นได้ว่าพัลซ์ทางด้านเอาท์พุทจะเกิดการผิดเพี้ยนทางด้านสัญญาณเมื่อเปรียบเทียบกับพัลซ์โดยทั่ว ๆ ไป เพราะว่าสัญญาณฮาร์มอนิกที่สูงกว่า ω_c หายไปนั่นเอง ถ้าผลการตอบสนองทางด้านเฟส $\phi(\omega)$ เป็นลิเนียร์ (linear) แล้วนั้น ผลที่เกิดตามมาก็คือความผิดเพี้ยนทางด้านเอาท์พุทนั้นจะน้อยลงจากผลตอบสนองทางด้านเฟส $\phi(\omega)$ ในรูปที่ 2.1 จะเห็นได้ว่ามีค่าความเป็นลิเนียร์ที่ย่าน $-\omega_c \le \omega \le +\omega_c$ ถ้าสัญญาณฮาร์มอนิกจะมีค่าน้อยที่สุดในช่วง ω_c แล้วนั้น ระบบจะเกิดความผิดเพี้ยนทางเฟสน้อยที่สุด

การหาเคอร์ฟทางด้านขนาดและเฟส ให้ $s = j\omega$ และ $H(j\omega)$ อยู่ในรูปโพล่าร์ฟอร์ม (polar form) สำหรับตัวอย่างผลตอบสนองทางด้านขนาดและเฟสของอัตราส่วนแรงดัน V_2/V_1 ของวงจร RC ในรูปที่ 2.2 ฟังก์ชันคือ

$$H(s) = \frac{V_2(s)}{V_1(s)} = \frac{1/RC}{S + 1/RC} \tag{2.2}$$

ให้ s = jω เราจะได้ว่า H(jω) คือ

$$H(j\omega) = \frac{1/RC}{j\omega + 1/RC} \tag{2.3}$$

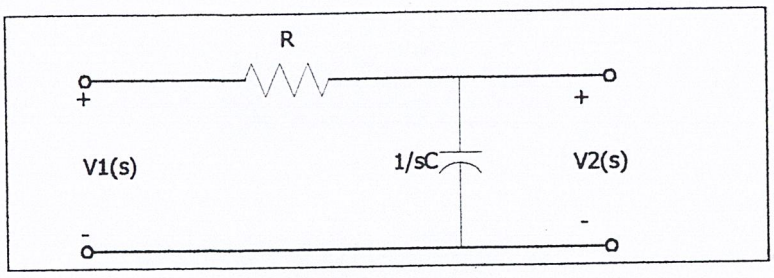


รูปที่ 2.1 แสดงผลตอบสนองทางด้านขนาดและเฟสของโลพาสฟิลเตอร์

สามารถเขียนให้อยู่ในรูปของโพลาร์ฟอร์มได้ดังนี้

$$H(j\omega) = \frac{1/RC}{\left(\omega^2 + 1/R^2C^2\right)^{1/2}} e^{-j\tan^{-1}\omega RC} = M(\omega)e^{j\phi(\omega)} \tag{2.4}$$

เคอร์ฟของแอมปลิจูดและเฟสสามารถพล็อตได้ดังรูปที่ 2.3 ที่จุด ω = 0 นั้น แอมปลิจูดมีค่าเป็น



รูปที่ 2.2 แสดงวงจร RC

1 และ เฟสจะมีค่าเป็น 0 องศา เมื่อ ω มีค่าเพิ่มขึ้นทั้งแอมปลิจูดและเฟสจะลดลง เมื่อ ω = 1/RC แล้วนั้นแอมปลิจูดจะมีค่าเท่ากับ 0.707 และ มีเฟสที่ 45 องศา ในจุดนี้เรียกว่า ฮาฟเพาเวอร์ (half-power) ของผลตอบสนองทางด้านขนาดและเฟส ถ้าค่า ω เข้าใกล้อินฟินิตี้แล้วค่าของ $M(\omega)$ จะมีค่าประมาณศูนย์ และ $\phi(\omega)$ จะมีค่าประมาณ 90 องศา

วิธีการหาค่าของผลตอบสนองทางด้านขนาดและเฟสจากโครงสร้างของโพล-ซีโร่ของฟังก์ชันต่อไปนี้

$$H(s) = \frac{A_0(s - z_0)(s - z_1)}{(s - p_0)(s - p_1)(s - p_2)} \tag{2.5}$$

เขียนในรูปของ $H(j\omega)$ ได้เป็น

$$H(j\omega) = \frac{A_0(j\omega - z_0)(j\omega - z_1)}{(j\omega - p_0)(j\omega - p_1)(j\omega - p_2)} \tag{2.6}$$

แต่ละเฟคเตอร์ (factor) $j\omega - Z_i$ หรือ $j\omega - P_j$ ซึ่งจะมีลักษณะเดียวกับรูปแบบทางด้านเวคเตอร์ (vector) ซึ่ง Z_i หรือ โพล P_j ที่พล็อตบนแกน $j\omega$ หรือแกนจินตภาพ (imaginary) เราสามารถแสดงเฟคเตอร์ในรูปทางโพล่าร์ฟอร์มได้ดังนี้

$$j\omega - z_i = N\, e^{j\omega_i},\ j\omega - p_j = M_j e^{j\theta_j} \tag{2.7}$$

ดังนั้นเราสามารถเขียนได้ดังนี้

$$H(j\omega) = \frac{A_0 N_0 N_1}{M_0 M_1 M_2} e^{j(\omega_0 + \omega_1 - \theta_1 - \theta_2 - \theta_3)} \tag{2.8}$$

ซึ่งได้แสดงไว้ในรูปที่ 2.4 ซึ่งค่าของ θ_1 มีค่าเป็นลบ

เราสามารถแสดงผลการตอบสนองทางด้านขนาด $M(\omega)$ อีกอย่างหนึ่งได้ดังนี้

$$M(\omega) = \frac{\prod_{i=0}^{n} \textit{แอมปลิจูดของเวคเตอร์ซีโร่บนเกนj}\omega}{\prod_{j=0}^{m} \textit{แอมปลิจูดของเวคเตอร์โพลบนเกนj}\omega}$$

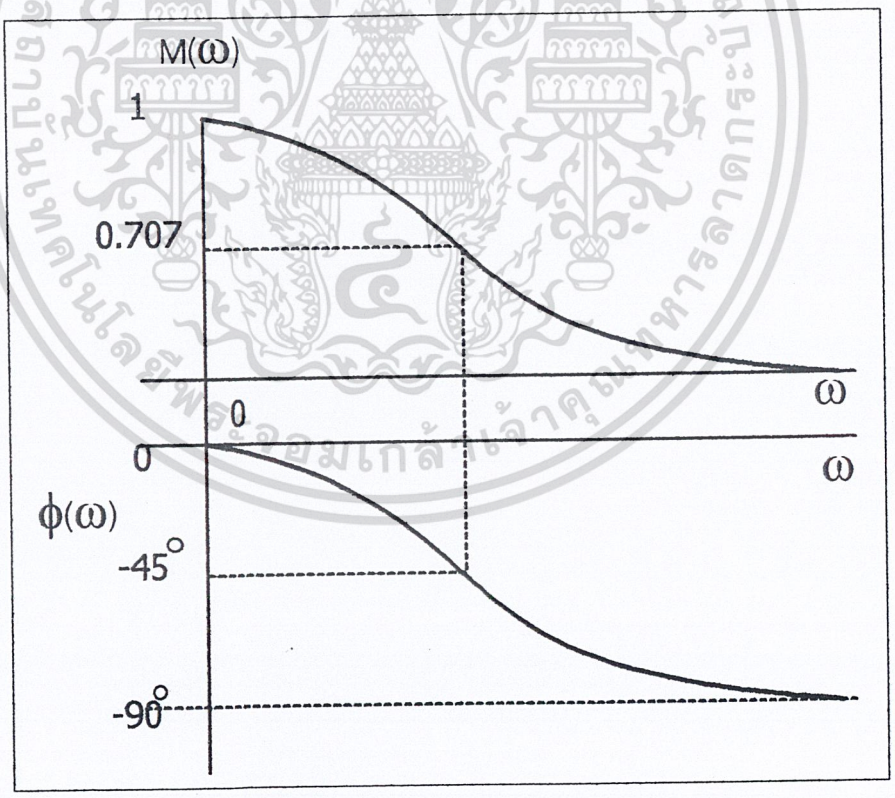
ในทำนองเดียวกันนั้นผลตอบสนองทางด้านเฟสสามารถเขียนได้ดังนี้

$$\phi(\omega) = \sum_{i=0}^{n} \textit{มุมของเวคเตอร์ซีโร่บนแกนj}\omega - \sum_{j=0}^{m} \textit{มุมของเวคเตอร์โพลบนแกนj}\omega$$

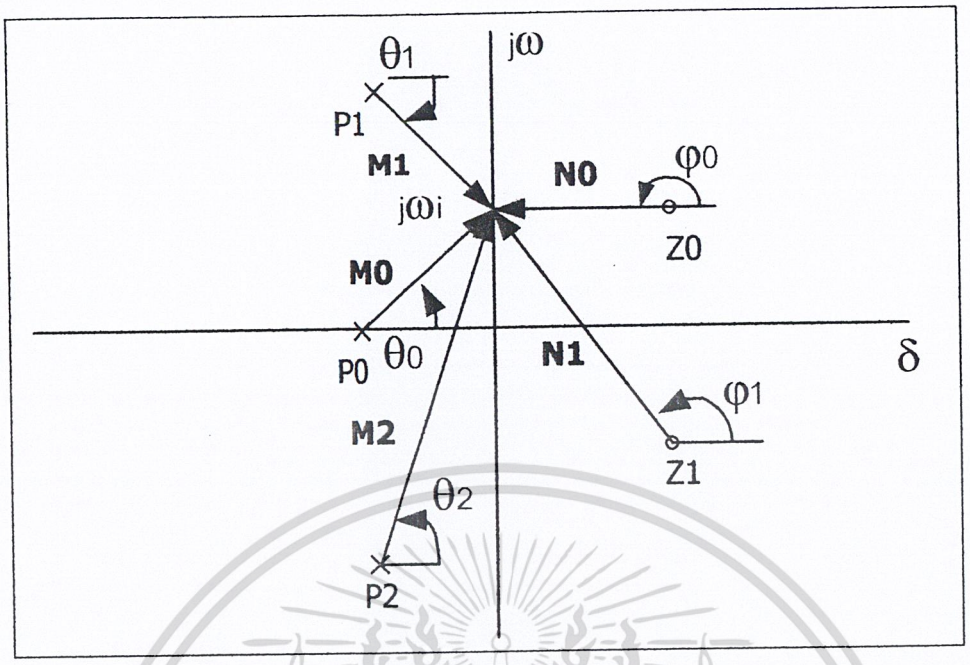
นี่คือความสัมพันธ์ของแอมปลิจูดและเฟสแบบจุดต่อจุด หรือในความหมายอีกอย่างหนึ่งคือ เราจะต้องวาดเวคเตอร์ฟอร์มของโพลและซีโร่ทุก ๆ จุดบนแกน jω เพื่อหาค่าของแอมปลิจูดและเฟส พิจารณาตัวอย่างต่อไปนี้

$$F(s) = \frac{4s}{s^2 + 2s + 2} = \frac{4s}{(s + 1 + j1)(s + 1 - j1)} \qquad (2.9)$$

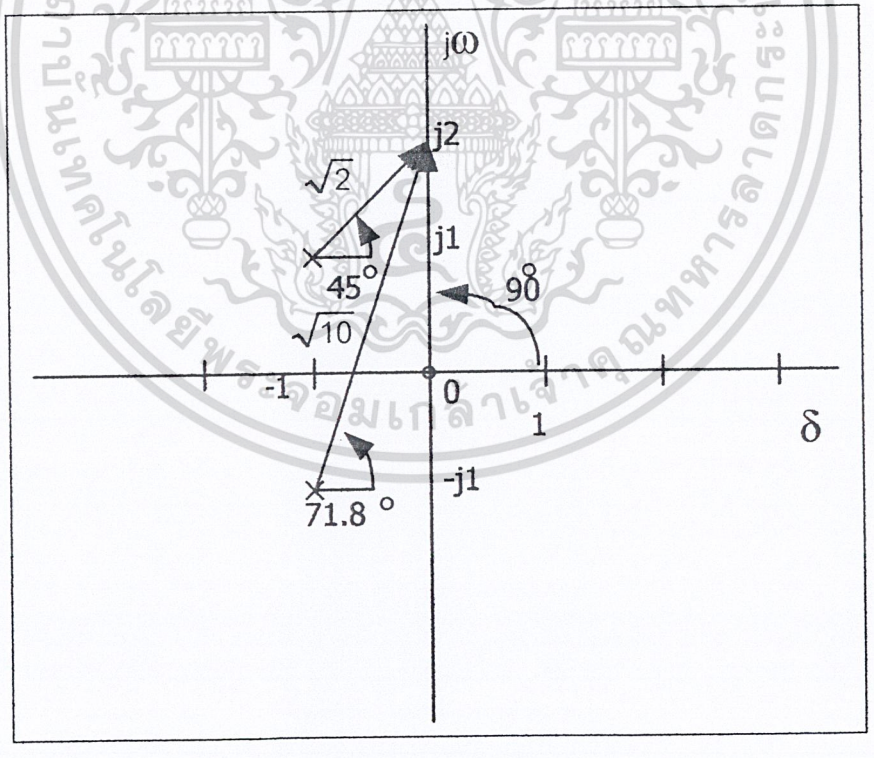
เราสามารถหาแอมปลิจูดและเฟสของ F(j2) จากโพลและซีโร่ของ F(s) โดยการวาดเวคเตอร์ในจุด ω = 2 ได้ดังรูปที่ 2.5 จากไดอะแกรมของโพลและซีโร่จะได้



รูปที่ 2.3 แสดงผลตอบสนองทางด้านขนาดและเฟสของวงจร RC



รูปที่ 2.4 แสดงการหาค่าของแอมปลิจูดและเฟสจากไดอะแกรมของโพล-ซีโร่



รูปที่ 2.5 แสดงการหาค่าแอมปลิจูดและเฟสจากไดอะแกรมของโพล-ซีโร่

$$M(j2) = 4\left(\frac{2}{\sqrt{2} \times \sqrt{10}}\right) = 1.78$$

และ $$\phi(j2) = 90^o - 45^o - 71.8^o = -26.8^o$$

ค่า M(j2) ,ϕ(j2), แอมปลิจูด และ เฟส ทั้ง 3 หรือ 4 จุด เราได้ข้อมูลสำหรับหาค่าผลตอบสนองทางด้านขนาดและเฟสอย่างคราว ๆ และที่ $\omega = 0$ เราจะเห็นได้ว่าเวคเตอร์ทางด้านขนาดของซีโร่จะอยู่ที่จุดกำเนิดมีค่าเป็นศูนย์

ดังนั้น M(j0) = 0 จากสมการที่ 2.9 นั้นเราสามารถเขียนได้เป็น

$$\lim_{\substack{\omega \to 0 \\ \omega > 0}} F(j\omega) = \frac{4(j0)}{(1 + j1)(1 - j1)} \qquad (2.10)$$

จากสมการนี้เราจะเห็นได้ว่าซีโร่ที่จุดกำเนิดนั้นจะมีค่า 90° ซึ่งเฟสนี้จะเลื่อนผ่านเวคเตอร์ทางด้านขนาดมีค่าเท่ากับ ศูนย์ จากรูปที่ 2.6a ซึ่งเฟสที่จุด $\omega = 0$ มีค่า $\phi(\omega) = 90^\circ - 45^\circ + 45^\circ = 90^\circ$ ณ.ความถี่สูงมาก ๆ ω_h นั้นค่า $\omega_h >> 1$ เวคเตอร์ทั้งหมดนั้นจะมีค่าเท่ากับ $\omega_h e^{j90^\circ}$ ซึ่งจะเห็นได้ดังรูปที่ 2.6b

$$M(\omega_h) \cong \frac{4\omega_h}{\omega_h^2} = \frac{4}{\omega_h}$$

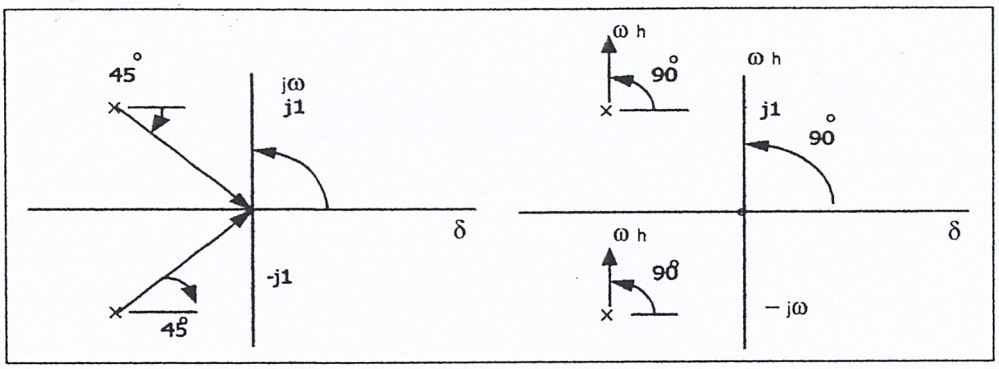
และ $$\phi(\omega_h) \cong 90^o - 90^o - 90^o = -90^o$$

ในการวิเคราะห์ในความถี่ต่าง ๆ ได้แสดงไว้ในตารางที่ 2.1 เราจะหาค่าของแอมปลิจูดและเฟสตามตาราง จากตารางนี้เราสามารถสเก็ตเคอร์ฟแอมปลิจูดและเฟสได้ตามรูปที่ 2.7

พิจารณาผลกระทบของโพล-ซีโร่บนแกน $j\omega$ ในผลตอบสนองความถี่ พิจารณาฟังก์ชันนี้ได้คือ

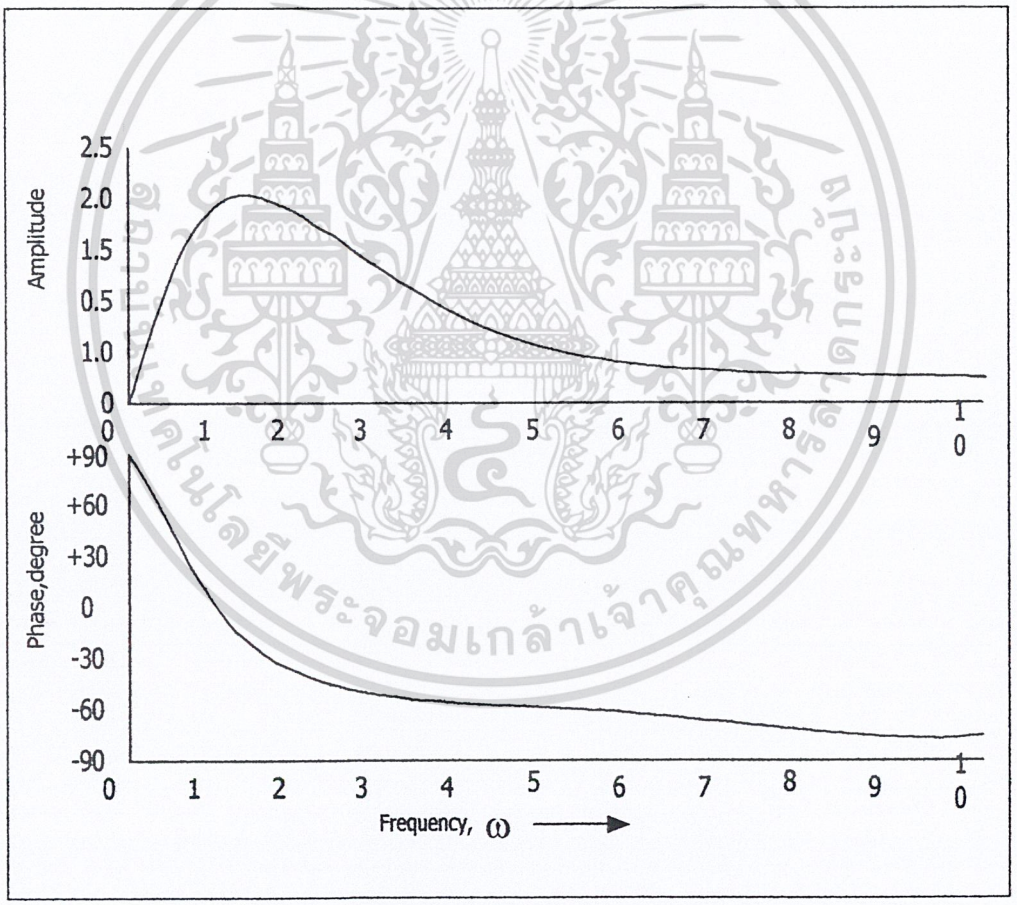
$$F(s) = \frac{s^2 + 1.03}{s^2 + 1.23} = \frac{(s + j1.015)(s - j1.015)}{(s + j1.109)(s - j1.109)} \qquad (2.11)$$

ในรูปที่ 2.8 แสดงไดอะแกรมของโพล-ซีโร่ ที่จุด $\omega = 1.015$ ซึ่งมีแอมปลิจูดเท่ากับศูนย์ ดังนั้นผลตอบสนองทางด้านขนาดจะมีค่าเท่ากับศูนย์ และที่จุด $\omega = 1.109$ เวคเตอร์ฟอร์มของโพลนั้นก็มีค่าแอมปลิจูดเท่ากับศูนย์ ดังนั้นผลตอบสนองทางด้านแอมปลิจูดที่จุดอินฟินิตี้จะมีค่าเท่ากับสมการที่ (2.11) ผลตอบสนองทางด้านเฟส เมื่อ $\omega < 1.015$ จากการพล็อตโพล-ซีโร่นั้นจะมีค่าเฟสเท่ากับ 0 เมื่อ $\omega > 1.015$ และ $\omega < 1.109$ และที่จุด $\omega = 1.015$ หรือ จุดที่อยู่สูงไปในขณะที่เวคเตอร์ฟอร์มของโพลและซีโร่อื่นนั้นจะมีลักษณะเดียวกันกับ $\omega < 1.015$ จะเห็นได้ว่าซีโร่



(a) (b)

รูปที่ 2.6 แสดงการหาค่าทางด้านขนาดและเฟสที่จุดศูนย์ และที่ความถี่สูงมาก ๆ



รูปที่ 2.7 แสดงผลตอบสนองทางด้านขนาดและเฟสของ F(s) ในสมการที่ (2.9)

ตารางที่ 2.1 แสดงเฟสและแอมปลิจูดของวงจร

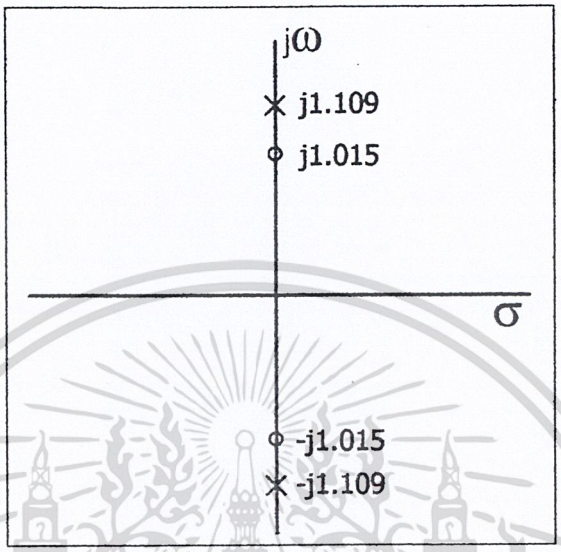
ความถี่ ω	แอมปลิจูด	เฟส,องศา
1.0	1.8	25.8
1.5	2.0	-5.3
3.0	1.3	-50.0
5.0	0.8	-66.0
10.0	0.4	-78.5

บนแกน $j\omega$ ผลตอบสนองทางด้านเฟสนั้นจะมีลักษณะที่ไม่ต่อเนื่องที่จุด $+180^\circ$ สำหรับความถี่ที่เพิ่มขึ้นก็เหมือนกับที่โพลบนแกน $j\omega$ และผลตอบสนองทางด้านเฟสจะไม่ต่อเนื่องที่ -180° ซึ่งในรูปที่ 2.11 นั้นแสดงถึงการพล็อตแอมปลิจูดและเฟสของ F(s) สำหรับย่านความถี่ $0.9 \leq \omega \leq 1.3$ แสดงไว้ในรูปที่ 2.9

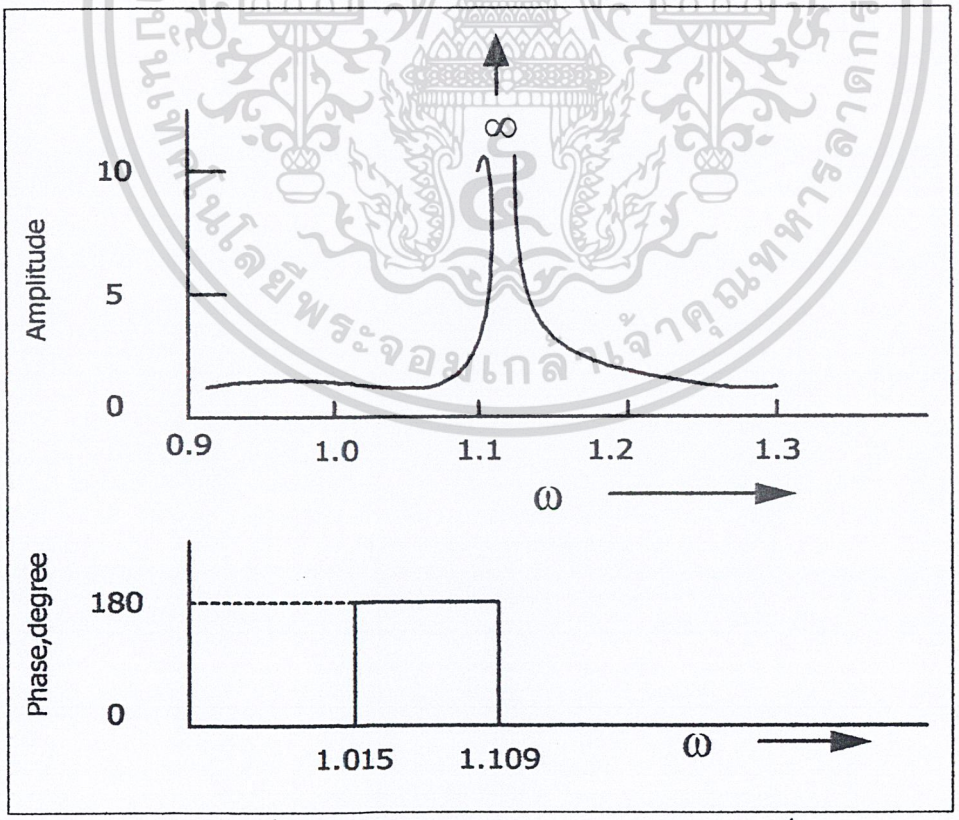
ในทางอุดมคติ เราจะเห็นได้ว่าถ้าเรามีซีโร่ที่ $z= -\sigma \pm j\omega_j$ เมื่อ σ คือ ค่าที่เล็กมาก ๆ เมื่อเทียบกับ ω_j แล้วจะมีการลาดเอียงของคุณสมบัติทางด้านขนาด ส่วนเฟสที่ใกล้เคียง ω_j นั้นแสดงไว้ดังรูปที่ 2.10 ในทำนองเดียวกันนั้น ถ้ามีโพลที่ $p = -\sigma \pm \omega_j$ เมื่อ σ มีค่าเล็กมาก ๆ นั้น แอมปลิจูดจะมีค่าสูงสุด และเฟสจะเริ่มลดลงอย่างรวดเร็วในขณะที่ใกล้ $\omega = \omega_j$ จะเห็นได้ดังรูปที่ 2.11 ในทางตรงกันข้ามนั้นเมื่อเรามีโพลและซีโร่ห่างจากแกน $j\omega$ มาก ๆ ซึ่งในที่นี้จะให้ค่า σ มีค่าสูงมาก ๆ เมื่อเทียบกับย่านความถี่ที่เราสนใจ จะเห็นได้ว่า โพลและซีโร่จะมีการผิดเพี้ยนน้อยมาก ในรูปของเคอร์ฟผลตอบสนองทางด้านขนาดและเฟสซึ่งจะมีผลกระทบต่อสเกลที่เพิ่มหรือลดเพื่อให้ครอบคลุมผลตอบสนองทางด้านขนาด

พิจารณาความมีเสถียรภาพ เรารู้ว่าจะต้องไม่มีโพลที่อยู่ทางด้านขวาบนระนาบ s-plane อย่างไรก็ตาม ทรานส์เฟอร์ฟังก์ชั่นจะต้องมีซีโร่อยู่ในทางด้านขวามือของระนาบพิจารณาไดอะแกรม ของโพลและซีโร่ในรูปที่ 2.12a และ รูปที่ 2.12b ซึ่งทั้งสองรูปนี้จะมีตำแหน่งของโพลที่เท่ากัน จะแตกต่างกันที่ซีโร่ซึ่งในรูป (a) จะอยู่ในระนาบซ้าย และที่ $s = -1 \pm j1$ ในขณะที่ซีโร่อีกรูปนั้นจะเป็นรูปกระจก (miror images) หรือมีค่าซีโร่ตรงกันข้ามกับรูป (a) และมีตำแหน่งที่ $s = 1 \pm j1$ ซึ่งผลตอบสนองทางด้านแอมปลิจูดทั้งสองรูปจะมีค่าที่เหมือนกันเพราะว่าความยาวของผลตอบ

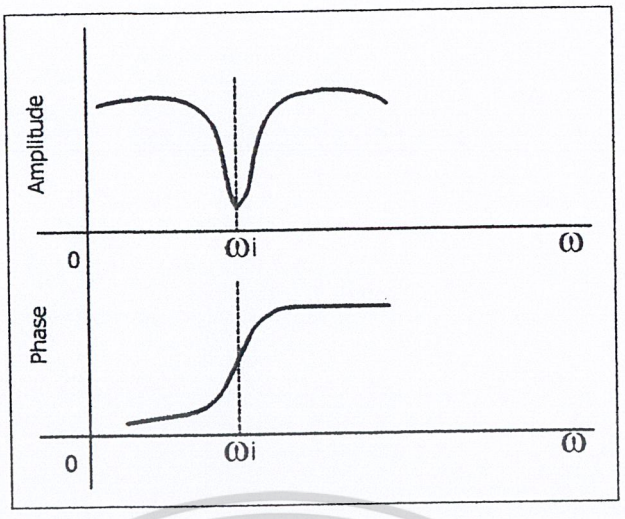
สนองของทั้งสภาวะมีค่าเท่ากัน ซึ่งจะเห็นได้ว่าค่าแอปโซลูทแมกนิจูด (absolute magenitude) ของเฟส (b) จะดีกว่า เฟสในรูป (a) เป็นเพราะว่าซีโร่อยู่ในระนาบทางด้านขวาจะเกิดการผิดเพี้ยน



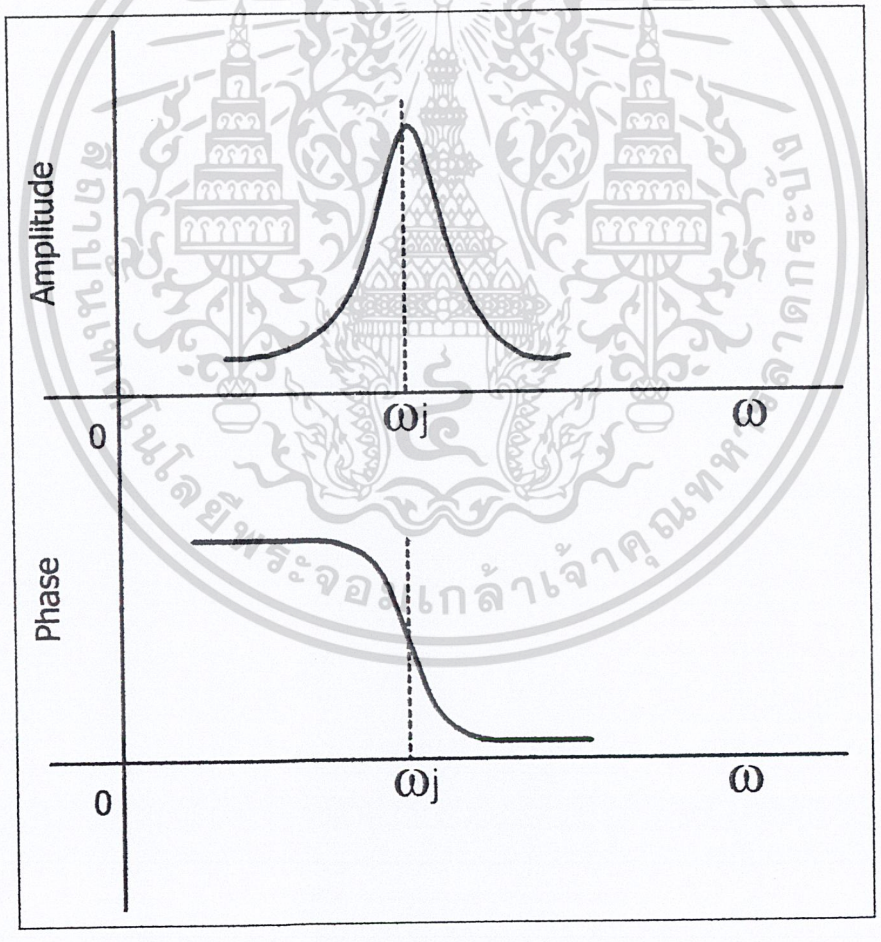
รูปที่ 2.8 แสดงตำแหน่งของโพลและซีโร่



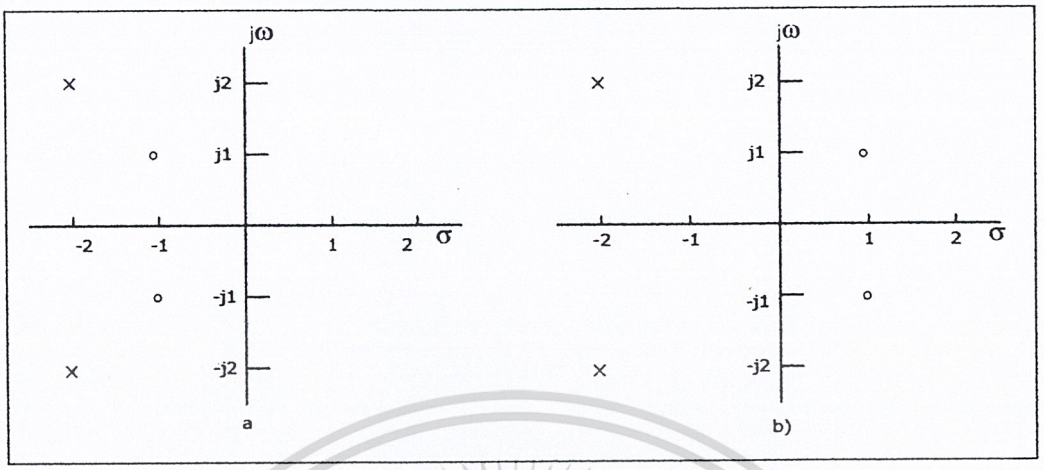
รูปที่ 2.9 แสดงแอมปลิจูดและเฟสของ F(s) ในรูปที่ 2.8



รูปที่ 2.10 แสดงผลกระทบของซีโร่เมื่ออยู่ใกล้แกน jω



รูปที่ 2.11 แสดงผลกระทบของโพลเมื่ออยู่ใกล้แกน jω



รูปที่ 2.12 แสดง (a) ฟังก์ชันเฟสมินิมั่ม (minimum phase function)
(b) ฟังก์ชันเฟสที่ไม่มินิมั่ม (nonminimum phase function)

การเลื่อนเฟสมากกว่า (บนพื้นฐานของแอปโซลูท แมกนิจูด) ซึ่งจะตรงกันข้ามกับส่วนในทางด้านซ้ายของระนาบ จากเหตุผลนี้เองเราจะมีข้อจำกัดตามมา ระบบฟังก์ชั่นกับซีโร่บนระนาบทางด้านซ้ายหรือบนแกน jω ซึ่งเรียกว่า มินิมั่มเฟสฟังก์ชั่น (minimum phase function) ถ้าฟังก์ชั่นมีซีโร่หนึ่งหรือมากกว่าอยู่ในระนาบด้านขวานั้นคือ ฟังก์ชันจะนอนมินิมั่ม(nonminimum) ในรูปที่ 2.13 จะเห็นได้ว่าผลตอบสนองทางด้านเฟสของฟังก์ชั่นมินิมั่มและไม่มินิมั่มแสดงได้ดังรูปที่ 2.12a และ 2.12b

พิจารณาไดอะแกรมของโพล-ซีโร่ในรูปที่ 2.14 จะเห็นได้ว่าซีโร่ในระนาบทางด้านขวาเป็นภาพกระจก (mirror images) ของโพลในระนาบทางด้านซ้าย ดังนั้นเวคเตอร์ที่วาดโพลนั้นคือ ω_1 บนแกน jω จะเห็นได้ว่าผลตอบสนองทางด้านขนาดจะต้องเป็นค่าคงที่ทุก ๆ ความถี่ และผลตอบสนองทางด้านเฟสจะต้องเป็นค่าคงที่ด้วย ในรูปที่ 2.15 แสดงเคอร์ฟผลตอบสนองทางด้านขนาด ส่วนในรูปที่ 2.15 แสดงไดอะแกรมของโพล-ซีโร่ ในระบบฟังก์ชั่นที่โพลอยู่ในระนาบทางด้านซ้ายและซีโร่เป็นภาพเงาของโพลบนแกน jω เรียกฟังก์ชันนี้ว่า ฟังก์ชันออพาส (all pass function) ซึ่งฟังก์ชันนี้ใช้ในการแก้ไขความผิดเพี้ยนทางด้านขนาดและเฟส

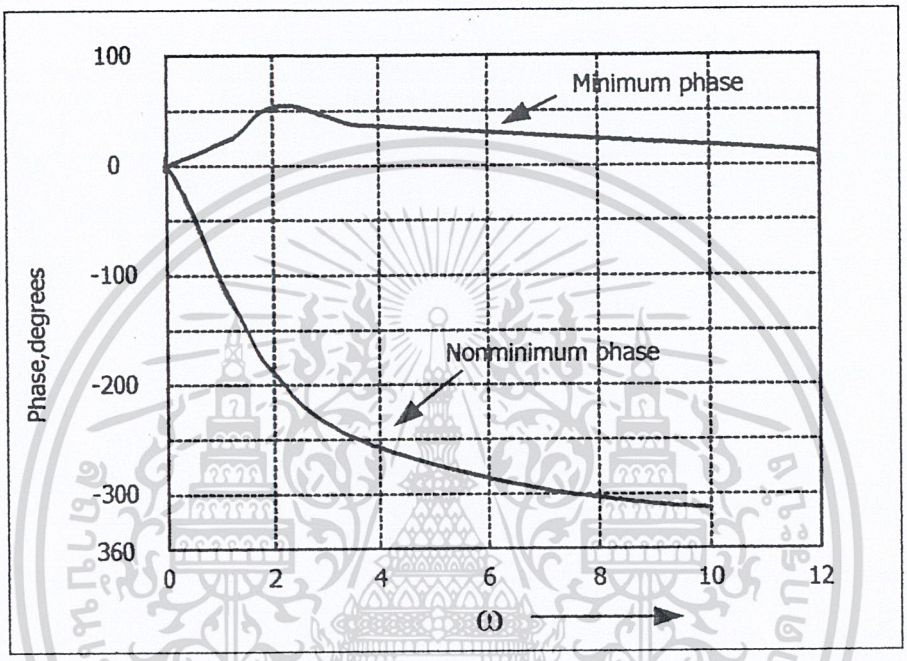
2.2 โบดพล็อต (bode plots)

ในการพล็อตกราฟล็อค เปรียบเทียบกับความถี่ทางด้านขนาดและเฟส การพล็อตนี้โดยทั่วไปเรียกว่า โบดพล็อต พิจารณาระบบฟังก์ชั่น

$$H(s) = \frac{N(s)}{D(s)} \qquad (2.12)$$

จะได้ผลตอบสนองทางด้านขนาดคือ

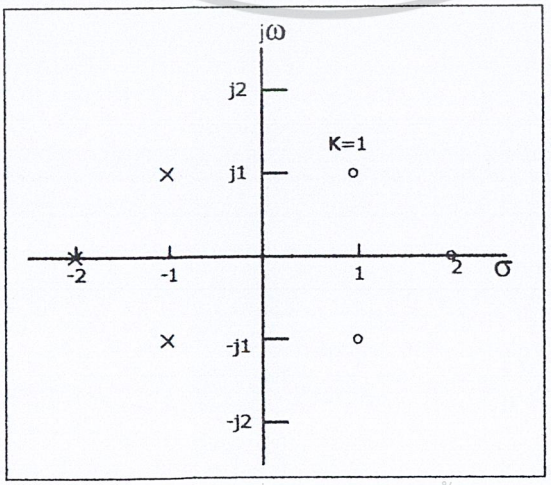
$$M(\omega) = |H(j\omega)| = \frac{|N(j\omega)|}{|D(j\omega)|} \qquad (2.13)$$



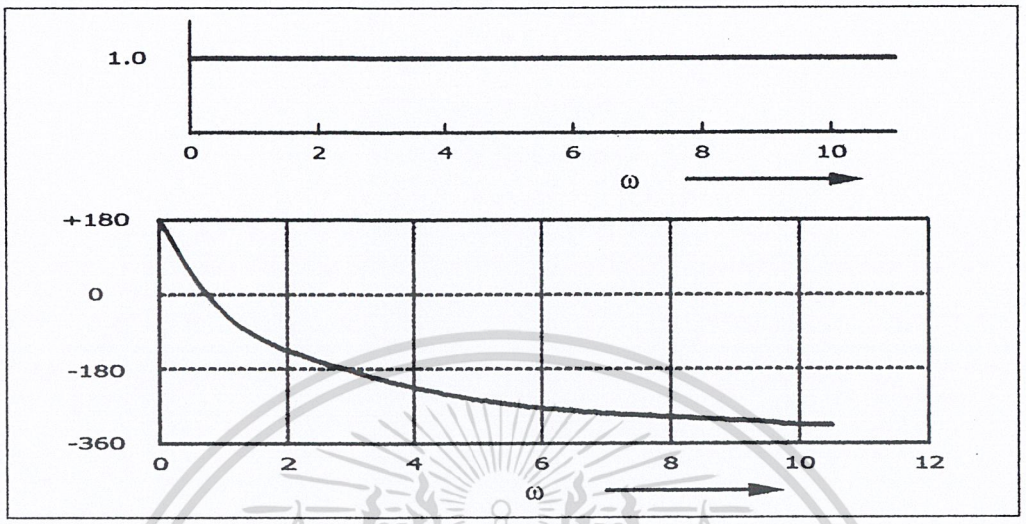
รูปที่ 2.13 แสดงการเปรียบเทียบฟังก์ชั่นทางเฟสระหว่างมินิมั่ม และนอนมินิมั่ม

ถ้าแสดงในเทอมของเดซิเบลจะได้ว่า

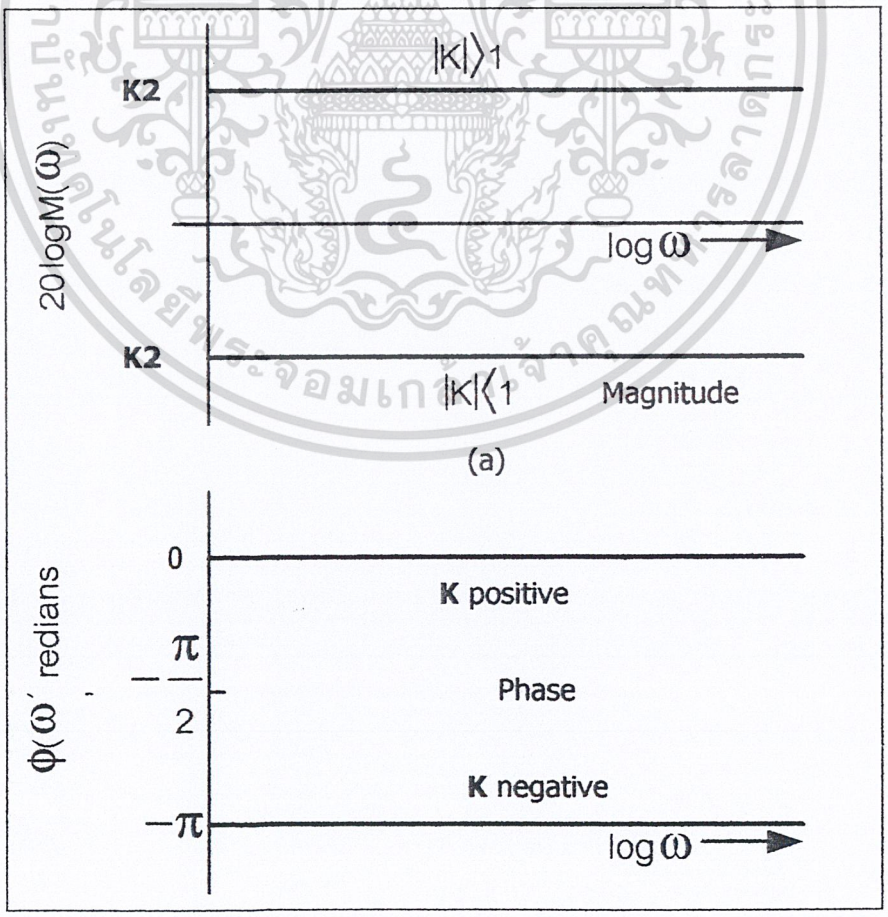
$$20\log M(\omega) = 20\log|N(j\omega)| - 20\log|D(j\omega)| \qquad (2.14)$$



รูปที่ 2.14 แสดงฟังก์ชันออพาส (all pass function)



รูปที่ 2.15 แสดงผลตอบสนองทางด้านเฟสและแอมปลิจูดของฟังก์ชั่นออพาสในรูปที่ 2.14



รูปที่ 2.16 แสดงค่าคงที่ของแอมปลิจูด และเฟส

ในเฟคเตอร์ฟอร์มทั้งของ N(s) และ D(s) จะมีอยู่ 4 ชนิดด้วยกันคือ

1. ค่าคงที่ K
2. ค่ารูท(root)ที่จุดกำเนิด
3. ค่ารูทจริง (real root) , $s + \sigma$
4. ค่าคอมเพล็กซ์ (complex) $s^2+2\alpha s+\alpha^2+\beta^2$

โดยทั่วไปแล้วในการพล็อตของล็อคแอมปิลจูด เราต้องการเพียงทดสอบผลตอบสนองทางด้านขนาดของเฟคเตอร์ทั้ง 4 ชนิดเท่านั้น ถ้าเฟคเตอร์ที่อยู่ในตัวเศษนั้นจะมีค่าเดซิเบลเป็นบวก ถ้าเฟคเตอร์เหล่านี้อยู่ในตัวส่วน แอมปลิจูดในหน่วยเดซิเบลจะมีสัญญาณเป็นลบ ต่อไปพิจารณาเงื่อนไขต่อไปนี้

1. เฟคเตอร์ K สำหรับค่าคงที่ K จะเท่ากับ

$$20\log K = K_2 \qquad (2.15)$$

ค่าคงที่ K_2 จะมีค่าเป็นลบ ถ้า $|K|<1$ และจะมีค่าเป็นบวกที่ $|K|>1$ ผลตอบสนองทางด้านเฟสจะมีค่าเป็น 0° หรือ 180° ซึ่งขึ้นอยู่กับค่าของ K มีค่าเป็นบวกหรือลบ โบดพล็อตจะแสดงค่าคงที่ของแอมปลิจูดและเฟสในรูปที่ 2.16

2. เฟคเตอร์ s เกนในระบบเดซิเบลจะเกี่ยวพันกับโพล(ซีโร่) ที่จุดกำเนิดคือ $\pm 20 \log \omega$ ดังนั้นการพล็อตแอมปลิจูดในหน่วยเดซิเบลเปรียบเทียบกับความถี่บนจุดโคออดิเนทบนกราฟล็อคคือเส้นตรงกับความชัน ± 20 เดซิเบลต่อดิเคด หรือ ± 6 เดซิเบลต่อออคทีฟ จากโบดพล็อตในรูป 2.17 จะเห็นได้ว่าจุดสูญเสียนั้นจะมีค่าเป็นศูนย์(ในหน่วยเดซิเบล) ซึ่งจะเกิดขึ้นที่ $\omega = 1$ และมีเฟสคือค่าคงที่สำหรับทุกค่าของ ω

3. เฟคเตอร์ $s+\alpha$ ให้ $\alpha = 1$ แล้วนั้น ค่าแอมปลิจูดก็คือ

$$\pm 20 iog|j\omega + 1| = \pm 20\log(\omega^2 + 1)^{1/2} \qquad (2.16)$$

ซึ่งแสดงไว้ในรูปที่ 2.18a ส่วนค่าของเฟสคือ

$$Arg(j\omega + 1)^{\pm 1} = \pm \tan^{-1}\omega \qquad (2.17)$$

ซึ่งแสดงไว้ในรูปที่ 2.18b

ส่วนการประมาณเส้นตรงของแอมปลิจูดแท้จริงนั้นเมื่อเปรียบเทียบกับเคอร์ฟซึ่งจะให้ผลทดสอบของเฟคเตอร์ $j\omega+1$ สำหรับ $\omega \leq 1$ นั้นจะได้ความถี่ต่ำคือ

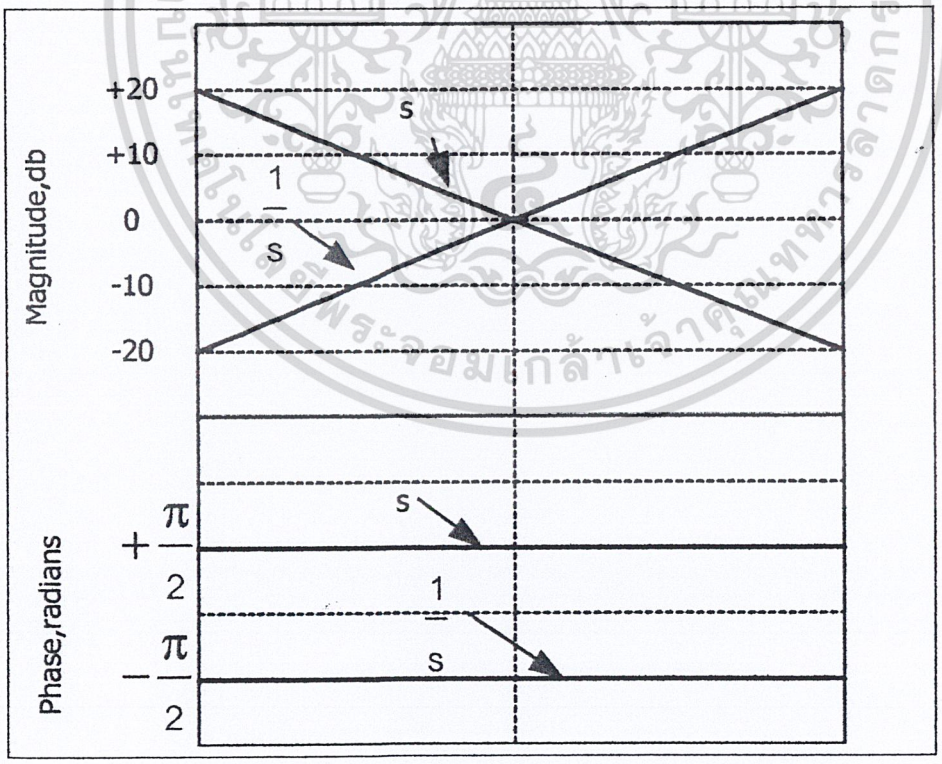
$$20\log|j\omega + 1|_{\omega \leq 1} \cong 20\log 1 = 0db \qquad (2.18)$$

สำหรับ $\omega \geq 1$ ที่ความถี่สูง

$$20 \log|j\omega + 1|_{\omega \geq 1} \cong 20 \log \omega \qquad (2.19)$$

ในรูปที่ (b) นั้นจะมีความชันคือ 20 dbต่อดิเคด หรือ 6 dbต่อออคทีฟ ซึ่ง ณ. จุดที่ $\omega = 1$ นั้นเรียกว่า ความถี่คัทออฟ (cutoff frequency) ซึ่งจะแสดงด้วยเส้นประในรูปที่ 2.18a ในตารางที่ 2.2 แสดงการเปรียบเทียบระหว่างค่าแอมปลิจูด ที่แท้จริงกับค่าประมาณ จะเห็นได้ว่าเกิดการผิดพลาดสูงสุดที่จุดความถี่คัทออฟหรือที่ $\omega = 1$

4. รูทคอมเพล็กซ์คอนจูเกท (Complex conjugate roots) สำหรับคอมเพล็กซ์คอนจูเกทนี้มีใช้มานานแล้วในระบบมาตราฐานของระบบจนกระทั่งเราสามารถใช้เคอร์ฟกันทั่วโลก เราสามารถอธิบายเกี่ยวกับโพลคอนจูเกท(ซีโร่) ในเทอมของแอมปลิจูด ω_0 และมุม θ จะวัดได้จากแกนลบจริงซึ่งแสดงไว้ในรูปที่ 2.19 ดังนั้นพารามิเตอร์เหล่านี้จะอธิบายถึงตำแหน่งโพล(ซีโร่) คือ ω_0 ซึ่งเรียกว่า ความถี่อันเดอร์แดมของออสซิลเลชั่น (underdamped frequency of oscillation) และ $\zeta = \cos\theta$ คือ เฟคเตอร์เดมมิ่ง (daming factor) ถ้าโพล-ซีโร่คู่ใดมีค่าจริงและจินตภาพ



รูปที่ 2.17 แสดงแอมปลิจูดและเฟสของโพลหรือซีโร่ที่จุด s=0

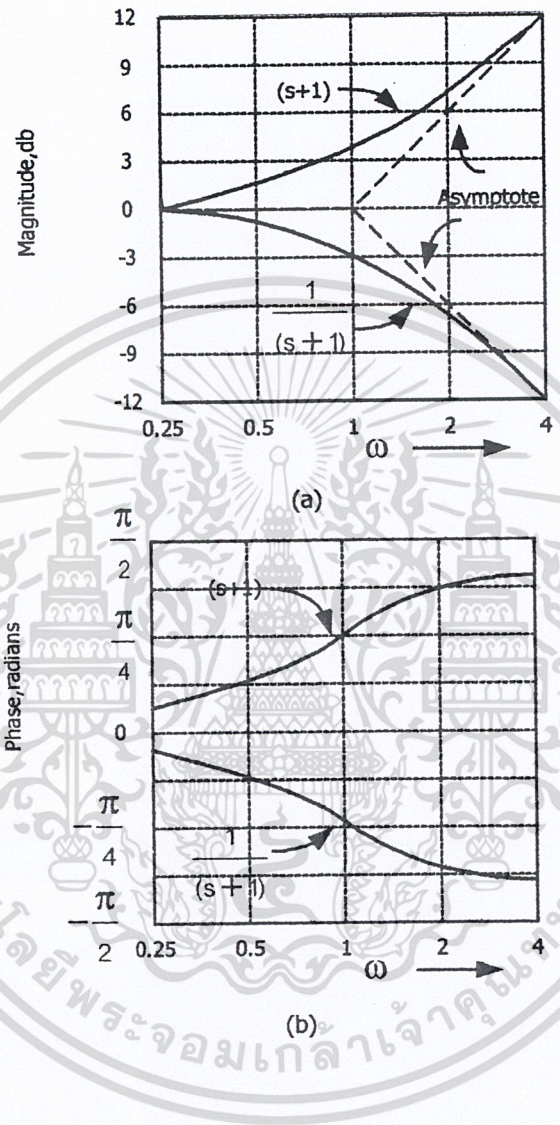
ตารางที่ 2 แสดงการเปรียบเทียบระหว่างค่าจริงและค่าที่ประมาณ

ความถี่	แอมปลิจูดที่แท้จริง(db)	ค่าประมาณของเส้นตรง(db)	ค่าผิดพลาด(db)
ω=1/4 ต่ำกว่า 2 ออคทีฟ	±0.3	0	±0.3
ω=1/2 ต่ำกว่า 1 ออคทีฟ	±1	0	±1
ω=1 ความถี่คัทออฟ	±3	0	±3
ω=2 สูงกว่า 1 ออคทีฟ	±7	±6	±1
ω=4 สูงกว่า 2 ออคทีฟ	±12.3	±12	±0.3

$$p_{1,2} = -\alpha \pm j\beta \tag{2.20}$$

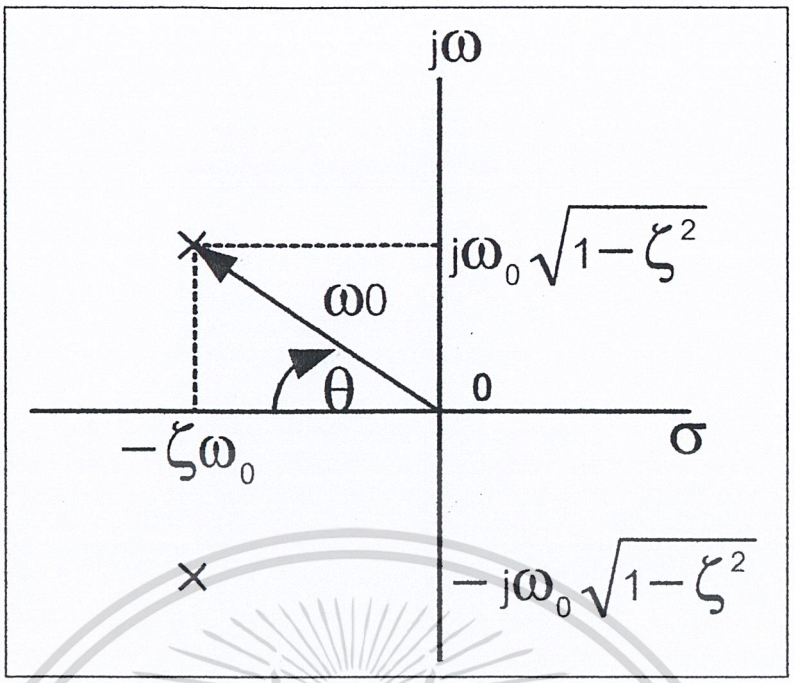
α และ β มีความสัมพันธ์กันกับ ζ และ ω_0 ดังนี้

$$\alpha = \omega_0 \cos\theta = \omega_0\zeta$$
$$\beta = \omega_0 \sin\theta = \omega_0\sqrt{1-\zeta^2} \tag{2.21}$$



รูปที่ 2.18 แสดงแอมปลิจูดและเฟสของโพลหรือซีโร่จำนวนจริงอย่างง่าย ๆ

จากคำจำกัดความของเฟคเตอร์เดมมิ่งนั้น $\zeta = \cos\theta$ จะเห็นได้ว่ามีค่าเข้าใกล้มุม θ ที่จุด $\pi/2$ นั้นคือค่าที่เล็กที่สุดของเฟคเตอร์เดมมิ่ง และเมื่อมุม θ เข้าใกล้ศูนย์องศา เฟคเตอร์เดมมิ่งจะมีค่าเข้าใกล้หนึ่ง



รูปที่ 2.19 แสดงตำแหน่งของโพลในเทอมของ ζ และ ω_0

ในการทดสอบโบดพล็อตสำหรับโพลคอนจูเกต(ซีโร่) เราให้ $\omega_0 = 1$ แล้วเราจะได้แอมปลิจูดคือ

$$\pm 20\log\left|1-\omega^2+j2\zeta\omega\right| = \pm 20\log\left[(1-\omega^2)^2+4\zeta^2\omega^2\right]^{1/2} \qquad (2.22)$$

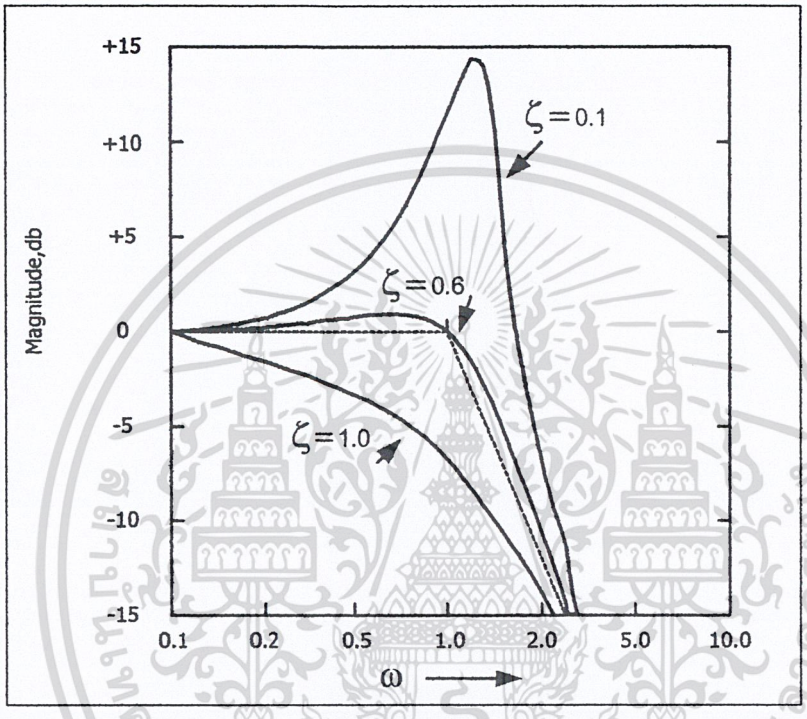
และ เฟสมีค่าเท่ากับ

$$\phi(\omega) = \tan^{-1}\frac{2\zeta\omega}{1-\omega^2} \qquad (2.23)$$

ถ้าทดสอบที่ความถี่สูงและต่ำจะได้ค่าแอมปลิจูดที่ความถี่ต่ำมีค่าเท่ากับ 0 เดซิเบล ที่ความถี่สูง(สำหรับ $\omega \geq 1$) มีค่าเท่ากับ $\pm 40\log\omega$ ซึ่งเส้นตรงนี้จะมีความชันเท่ากับ 40 เดซิเบลต่อดิเคท หรือ 12 เดซิเบลต่อออคทีฟ ดังนั้นเฟคเตอร์เดมมิ่ง ζ เป็นตัวสำคัญที่จะทำให้เข้าใกล้เส้นตรงที่ได้ประมาณไว้ ในรูปที่ 2.20 แสดงการประมาณค่าเส้นตรงของโพลคอนจูเกต (ω_0=1) ซึ่งจะแสดงด้วยเส้นประ ส่วนเคอร์ฟอื่น ๆ นั้นจะแสดงด้วยเส้นทึบ โดยทั่ว ๆ ไปแล้วนั้นฟังก์ชั่นความถี่นอมอลไลซ์ (freqency normalized function)จะทำการพล็อตทั้งขนาดและเฟสดังรูปที่ 2.21 และ 2.22

$$G(s) = \frac{1}{(s/\omega_0)^2 + 2\zeta(s/\omega_0)+1} \qquad (2.24)$$

จะเห็นได้ว่าผลการตอบสนองทางด้านเฟสจะแสดงได้จากสเกลของล็อกซึ่งเป็นฟังก์ชั่นที่มีค่า $\omega/\omega_0 = 1$ เฟสจะอยู่ที่ $\omega = \omega_0$ มีค่าเท่ากับ 90° หรือ $\pi/2$ เรเดียน ส่วนในการคอนจูเกตของซีโร่ก็เพียงแต่กลับสัญญาณบนสเกลของเคอร์ฟทางด้านแอมปลิจูดและเฟส

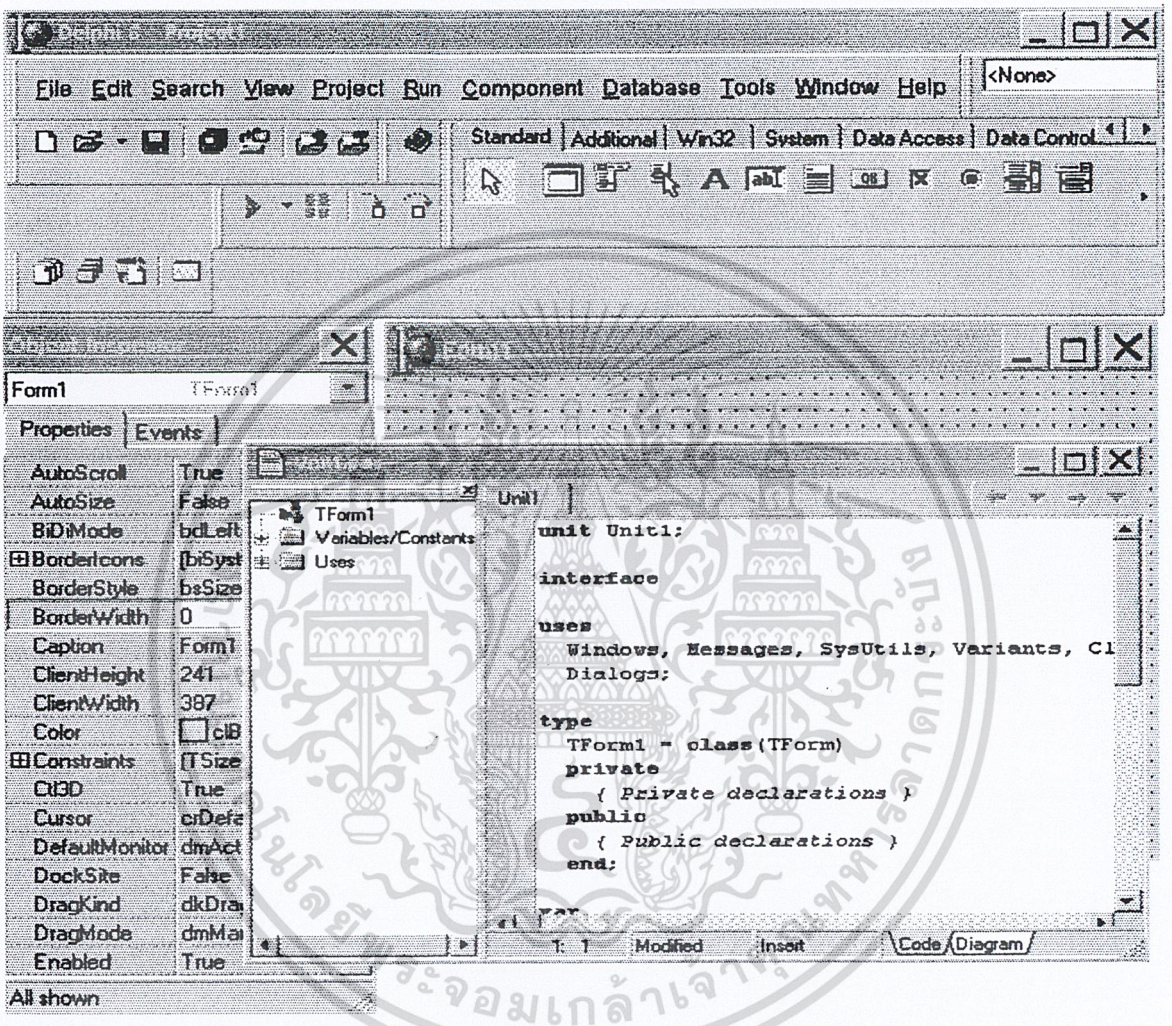


รูปที่ 2.20 แสดงการเปรียบเทียบแอมปลิจูดความถี่ของโพลอันดับสอง

บทที่ 3

แนะนำโปรแกรมเดลไฟ (DELPHI)

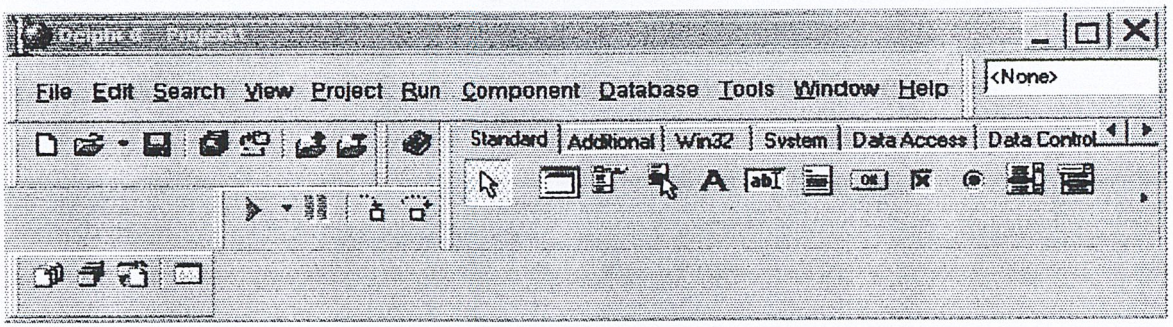
เมื่อเรียกใช้โปรแกรม Delphi จะปรากฏจอภาพดังรูป ซึ่งประกอบไปด้วยหน้าต่าง 4 หน้าต่างดังนี้



รูปที่ 3.1 แสดงหน้าต่างเมื่อเรียกใช้โปรแกรม Delphi

หน้าต่างหลัก (Main Window)

เป็นหน้าต่างแรกที่ปรากฏเมื่อเรียกใช้โปรแกรม Delphi ซึ่งประกอบไปด้วย 3 ส่วนคือ Menubar, Speedbar และ Component Palette โดยมีรายละเอียดดังนี้



รูปที่ 3.2 แสดงลักษณะของหน้าต่างหลัก

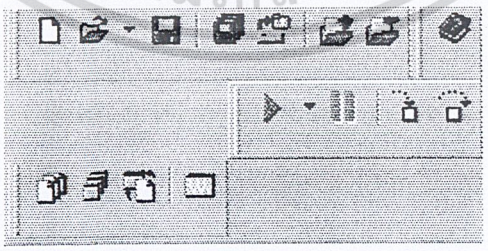
เมนู (Menubar)

เมนู เป็นส่วนหลักที่ใช้ในการทำงานคำสั่งต่าง ๆ เกือบทั้งหมดสามารถเรียกใช้ผ่านเมนูได้โดยมีลักษณะดังรูป

รูปที่ 3.3 แสดงลักษณะของเมนู

Speedbar

Speedbar ประกอบไปด้วยปุ่มของคำสั่งที่ใช้งานบ่อย ๆ สำหรับการทำงานของ Delphi สามารถใช้การ Click Mouse ที่ปุ่ม เพื่อเรียกคำสั่งเหล่านั้นมาทำงาน โดยปกติแล้ว Speedbar จะอยู่ใต้ Menubar และอยู่ด้านซ้ายของจอ ซึ่งมีลักษณะดังรูป



รูปที่ 3.4 แสดงลักษณะของ Speedbar

Component Palette

ภายใน Component Palette จะมีรูปเล็กๆ ที่เรียกว่าไอคอน ที่แสดงถึงคอมโพเนนต์ต่าง ๆ ใน Delphi โดยจะจัดแบ่งออกเป็นหลายกลุ่ม วางอยู่ในแท็ปต่าง ๆ ของ Component Palette สามารถ Click ที่ชื่อแท็ปเพื่อแสดงไอคอนที่จัดอยู่ในกลุ่มนั้น ๆ แต่ละไอคอนจะแสดง Click บนฟอร์มในตำแหน่งที่ต้องการวางคอมโพเนนต์เหล่านั้น

รูปที่ 3.5 แสดงลักษณะ Component Palette

หน้าต่าง Object Inspector

Object Inspector เป็นที่สำหรับกำหนดคุณสมบัติและโพรซีเดอร์ที่ควบคุม Event ของ Component ในฟอร์มด้านบนของ Object Inspector จะเป็นคอมโบบ็อกที่แสดงชื่อของคอมโพเนนต์ และชนิดของคอมโพเนนต์ เช่น ถ้าเลือกฟอร์ม คอมโบบ็อกจะแสดง "form 1: Tform 1" โดยที่ Form 1 เป็นชื่อของฟอร์มที่เราเลือกและ Tform เป็นชื่อชนิดของคอมโพเนนต์ (Delphi จะใส่ T ไว้ที่หน้าชื่อชนิดของคอมโพเนนต์ที่เรียกใช้กันทั่วไป เช่น TForm, Tbutton ซึ่งจะแทนชนิดของคอมโพเนนต์นั้น ๆ)

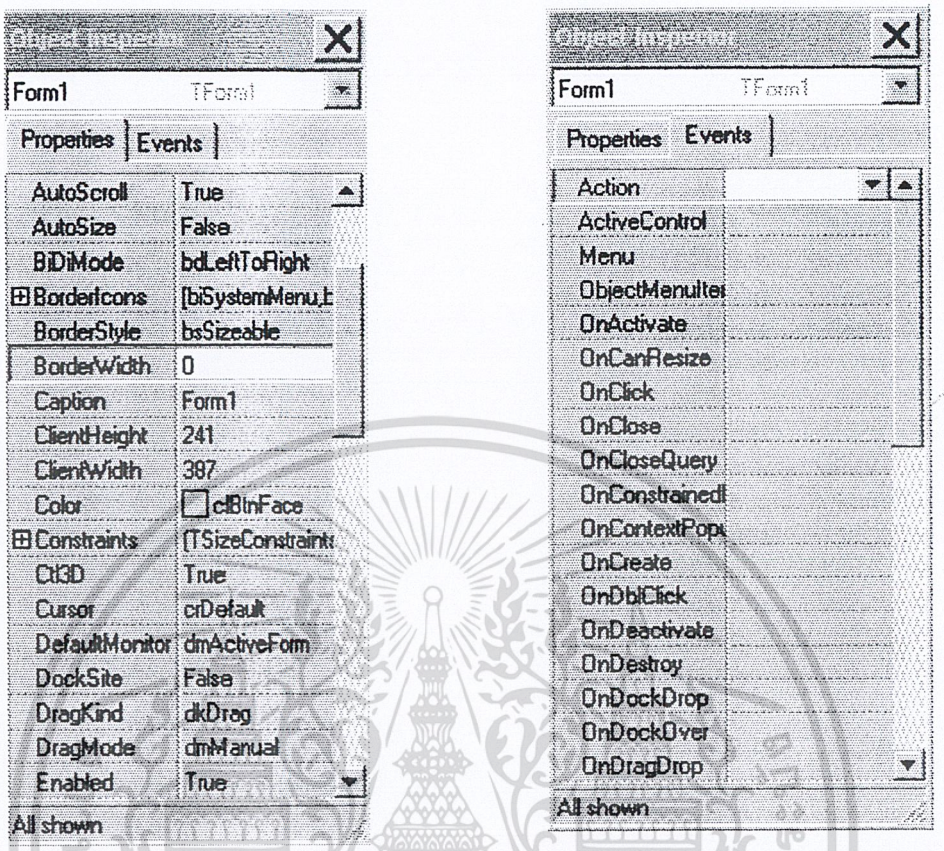
ภายใน Object Inspector จะประกอบไปด้วยแท็ปสองแท็ปดังต่อไปนี้

แท็ป Properties

จะแสดงคุณสมบัติ (Properties) ต่าง ๆ ของ Component ภายในแท็ป Properties จะประกอบไปด้วยสองคอลัมน์ โดยที่ทางด้านซ้ายจะเป็นชื่อของคุณสมบัติ และทางด้านขวาจะเป็นค่าของคุณสมบัตินั้น ๆ

แท็ป Events

ใช้กำหนดคำสั่งต่าง ๆ ที่จะกระทำก็ต่อเมื่อเกิดเหตุการณ์หนึ่งขึ้นกับคอมโพเนนต์ เช่น ในการ Click Mouse หรือกดปุ่ม <Enter> ที่คอมโพเนนต์นั้นก็จะเกิดการทำงานของอีเวนต์ OnClick ขึ้น



รูปที่ 3.6 แสดงลักษณะของหน้าต่าง Object Inspector

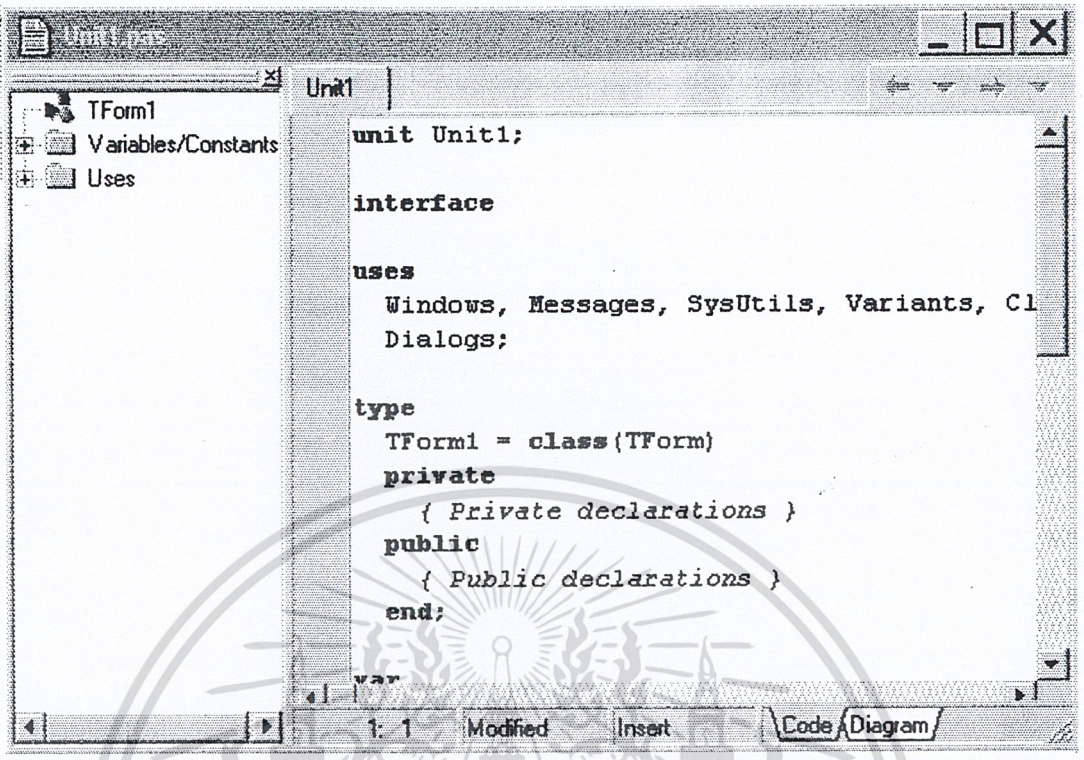
หน้าต่าง Form Designer

Form Designer ใช้สำหรับออกแบบส่วนที่ติดต่อกับผู้ใช้ ซึ่งสามารถนำคอมโพเนนต์ต่าง ๆ จาก Component Palette มาวางลงบน Form Designer โดยที่เราสามารถแก้ไขขนาด ย้ายคอมโพเนนต์ไปมา รวมทั้งเพิ่มหรือลบคอมโพเนนต์ออกจากฟอร์มได้ โดยสามารถเปรียบ Form Designer ได้กับกระดาษวาดเขียนที่สามารถเขียนหรือลบส่วนที่วาดได้

ทุกครั้งที่สร้างโปรเจกต์ใหม่จะมีฟอร์มใหม่เกิดขึ้นชื่อ Form1 ซึ่งเป็นฟอร์มเริ่มต้น ที่สามารถเริ่มพัฒนาแอปพลิเคชั่นได้จากฟอร์มนี้

หน้าต่าง Code Editor

Code Editor เป็น Text Editor สำหรับการเขียนโปรแกรมใน Delphi ซึ่งประกอบไปด้วยคุณสมบัติต่าง ๆ ที่อำนวยความสะดวกในการเขียนโปรแกรม Delphi ดังนี้



รูปที่ 3.7 แสดงลักษณะของหน้าต่าง Code Editor

ส่วนล่างของ Code Editor จะแสดงสถานะการทำงาน ซึ่งประกอบไปด้วย

- ตำแหน่งของเคอเซอร์ บอกว่าตอนนี้เคอเซอร์อยู่ที่บรรทัดไหน และคอลัมน์ที่เท่าไร

- สถานะการแก้ไข เป็นตัวบอกว่าไฟล์นั้นได้มีการแก้ไขแล้วหรือยัง ถ้ามีการแก้ไขแล้วจะปรากฏคำว่า Modified

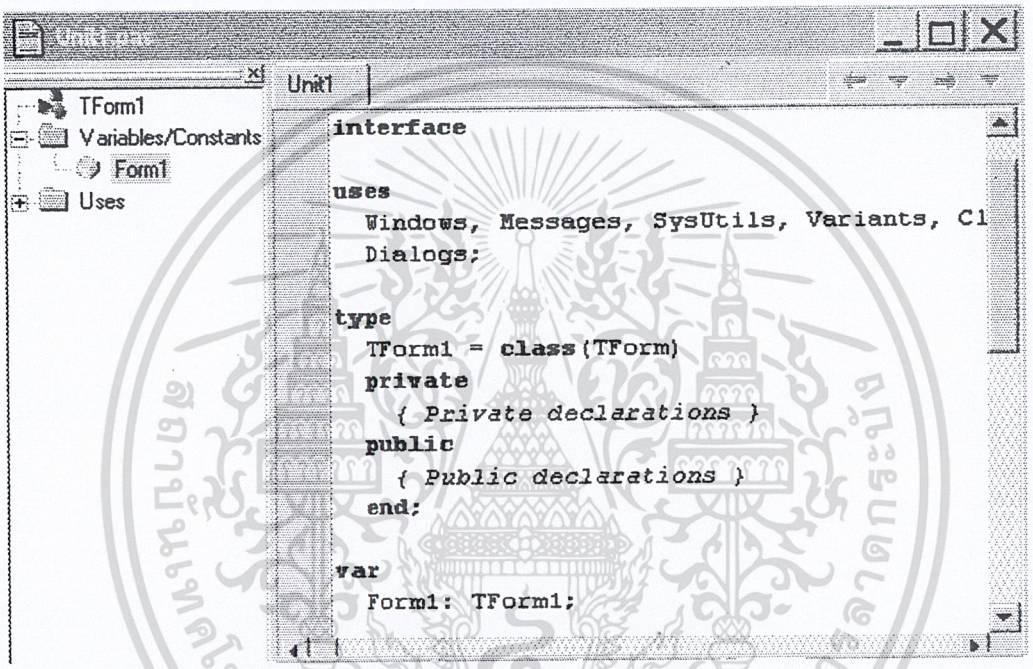
- สถานะการพิมพ์ เป็นตัวบอกสถานะการพิมพ์ว่าเป็นการพิมพ์แทรก (insert) หรือพิมพ์ทับ (Overwrite)

เมื่อเปิด Delphi ขึ้นมาครั้งแรก ภายใน Code Editor จะแสดง Code เริ่มต้นที่ Delphi สร้างเอาไว้ให้ชื่อ Unit1 ภายใน Code Editor เราสามารถเปิดไฟล์ได้พร้อมกันหลายไฟล์ โดยสามารถเลือกไฟล์ที่ต้องการแก้ไขได้จากแท็ปด้านบนของ Code Editor โดย Click บนแท็ปที่แสดงชื่อไฟล์ที่ต้องการ

Code Editor ช่วยเพิ่มความสะดวกในการเขียนโปรแกรมเป็นอย่างมาก สามารถปรับแต่งคุณสมบัติต่าง ๆ ของ Code Editor ได้ โดยเลือกเมนู Tool>Environment Options

Code Explorer

ภายในหน้าต่าง Code Editor ประกอบได้ด้วยหน้าต่างซ้อนอยู่ 2 หน้าต่าง เรียกหน้าต่างที่อยู่ทางด้านซ้ายว่า "Code Explorer" เป็นหน้าต่างที่มีลักษณะเป็น Tree เพื่อช่วยในการเข้าถึง Objet ต่าง ๆ ที่กำหนดภายใน Unit เช่น ตัวแปร, Method, Procedure ซึ่งมีวิธีการใช้คือ เมื่อเลือก Object ที่อยู่ใน Code Explorer แล้วตำแหน่งของเคอเซอร์ที่อยู่ใน Code Editor จะเคลื่อนที่ไปยังตำแน่งที่ Object นั้นจะปรากฏอยู่ เช่น เมื่อเลือกอ็อปเจกต์ที่เป็นชื่อตัวแปรใน Code Editor จะกระโดดไปยังบรรทัดที่มีการประกาศตัวแปรนั้น



รูปที่ 3.8 แสดงลักษณะของ Code Explorer

สามารถแก้ไขคุณสมบัติต่าง ๆ ของ Code Explorer ได้ที่ Tools>Environment Option ที่แท็ปชื่อ Explorer

การทำงานและการจัดการกับโปรเจกต์

โปรเจกต์ (Project) คือไฟล์ที่ใช้สำหรับเก็บรวบรวม ฟอร์ม, Unit ต่าง ๆ เข้าไว้ด้วยกันเพื่อใช้ในการสร้างแอปพลิเคชั่น เมื่อทำการคอมไพล์โปรเจกต์ ก็จะได้แอปพลิเคชั่น 1 ตัว

ในการทำงานกับโปรเจกต์ใน Delphi สามารถเรียกคำสั่งต่าง ๆ จากเมนูหลักได้ทั้งหมด

การสร้างแอปพลิเคชั่นใหม่ (New Application)

เมื่อทำการสร้างแอปพลิเคชั่นขึ้นมาใหม่ Delphi จะทำการสร้างโปรเจกต์ใหม่ให้ 1 โปรเจต์ ซึ่งประกอบด้วยฟอร์มพื้นฐาน 1 ฟอร์มและ Unit ที่ใช้สำหรับฟอร์มนั้น 1 Unit โดยสามารถสร้างแอปฟลิเคชั่นใหม่ได้โดยเลือกที่เมนู File>New Application

แอปพลิเคชั่นใหม่ที่สร้างขึ้นมานี้ สามารถรันได้ทันทีแม้ว่ายังไม่ได้วางคอมโพเนนต์ใด ๆ เลย สามารถทดสอบการรันแอปพลิเคชั่นใหม่นี้ได้โดยการ Click ที่เมนู Run>Run

แอปพลิเคชั่นที่ได้จากการรัน จะเป็นหน้าต่างว่าง ๆ เนื่องจากเรายังไม่ได้ทำการวางคอมโพเนนต์ใด ๆ ลงบนฟอร์มเลย สามารถปิดแอปพลิเคชั่นที่รันอยู่ได้โดย Click ที่ปุ่มกากบาท

รูปที่ 3.9 แสดง From ของการ Run

การบันทึกโปรเจกต์

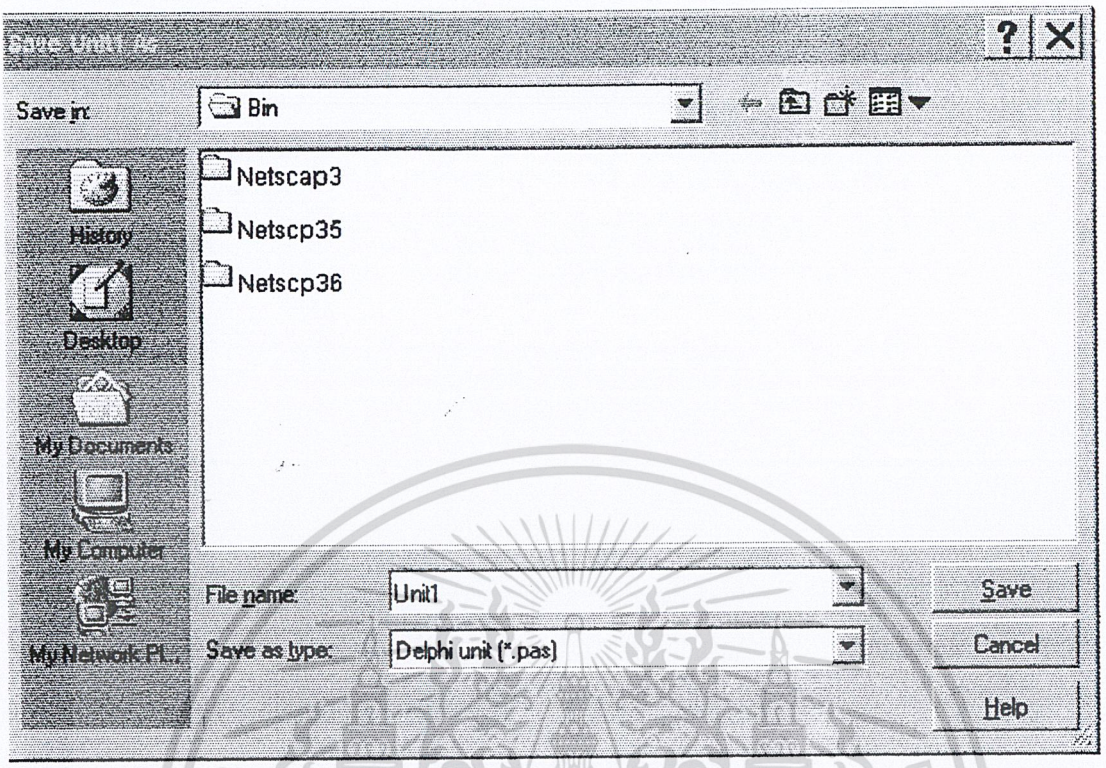
การบันทึกข้อมูลใน Delphi สามารถทำได้ 4 ลักษณะได้แก่

Save

เป็นการสั่งให้บันทำข้อมูล โดยถ้าเป็นการบันทึกครั้งแรก Delphi จะมีไดอะล็อกขึ้นมาถามว่าต้องการบันทึกข้อมูลด้วยไฟล์ชื่ออะไร และสำหรับในกรณีที่ต้องการบันทึกข้อมูลที่เปิดขึ้นมาแก้ไข จะเป็นการใช้ชื่อไฟล์เดิมในการบันทึกข้อมูล สามารถทำการสั่งให้บันทึกข้อมูลด้วยวิธีการนี้โดย ให้ Click ที่เมนู File>Save

Save As

จะเป็นการบันทึกไฟล์ที่ถูกแก้ไขด้วยชื่อไฟล์ใหม่ โดยที่จะมีหน้าต่างไดอะล็อก ให้ใส่ชื่อไฟล์ที่ต้องการจะเก็บข้อมูล โดยจะมีขั้นตอนดังนี้



รูปที่ 3.10 แสดงการ save as โปรแกรม Delphi

Save Project As

จะเป็นการบันทึกเฉพาะไฟล์โปรเจกต์ด้วยชื่อใหม่ ซึ่งไฟล์ที่ได้จะมีนามสกุลเป็น .DPR

Save All

จะเป็นการบันทึกข้อมูลทั้งหมดที่ถูกแก้ไขในโปรเจกต์ที่กำลังทำงานอยู่ในขณะนั้น จะมีขั้นตอนเหมือนกับการใช้คำสั่ง Save กับทุกไฟล์แล้วตามด้วยคำสั่ง Save Project As โดยถ้าไฟล์ใดมีการตั้งชื่อไฟล์แล้วจะใช้ชื่อไฟล์เดิม แต่ถ้ายังไม่มีการตั้งชื่อไฟล์ ก็จะปรากฏไดอะล็อกขึ้นมาถามชื่อว่าจะชื่ออะไร โดย Delphi จะถามชื่อของ Unit และชื่อของโปรเจกต์ตามลำดับชื่อ Unit คือชื่อของไฟล์ .PAS ที่เก็บโปรแกรมที่เขียนขึ้น และชื่อโปรเจกต์คือชื่อไฟล์ .DPR ซึ่งเป็นตัวเก็บว่าในโปรเจกต์จะประกอบด้วยฟอร์ม และ Unit อะไรบ้าง

ชื่อของ Unit และชื่อของโปรเจกต์จะต้องเป็นชื่อที่ไม่ซ้ำกัน Delphi ไม่อนุญาตให้ใช้ชื่อ Unit กับชื่อโปรเจกต์เป็นชื่อเดียวกัน

การเพิ่มฟอร์มเข้าไปในโปรเจกต์

เนื่องจากในปกติแอปพลิเคชั่นที่สร้างขึ้นมา จะประกอบด้วยฟอร์มมากกว่า 1 ฟอร์ม ดังนั้น Delphi จึงอนุญาตให้สามารถเพิ่มฟอร์มเข้าไปในโปรเจกต์ได้ ซึ่งมี 2 วิธีคือ

- การสร้างฟอร์มขึ้นใหม่
- การนำฟอร์มที่มีอยู่แล้วมาใช้

โดยมีรายละเอียดดังต่อไปนี้

1. **การสร้างฟอร์มใหม่** (New Form)

สามารถสร้างฟอร์มใหม่ได้โดย Click ที่เมนู File>New Form ฟอร์มใหม่ที่ได้จะเป็นฟอร์มเปล่า ๆ เช่นเดียวกับ ฟอร์มเริ่มต้องตอนสร้างโปรเจกต์ใหม่ เมื่อเราทำการสร้างฟอร์มใหม่ สิ่งที่ได้จะประกอบไปด้วยฟอร์มเปล่า ๆ 1 ฟอร์มกับ Unit

2. **การนำฟอร์มที่มีอยู่แล้วมาใช้ในโปรเจกต์**

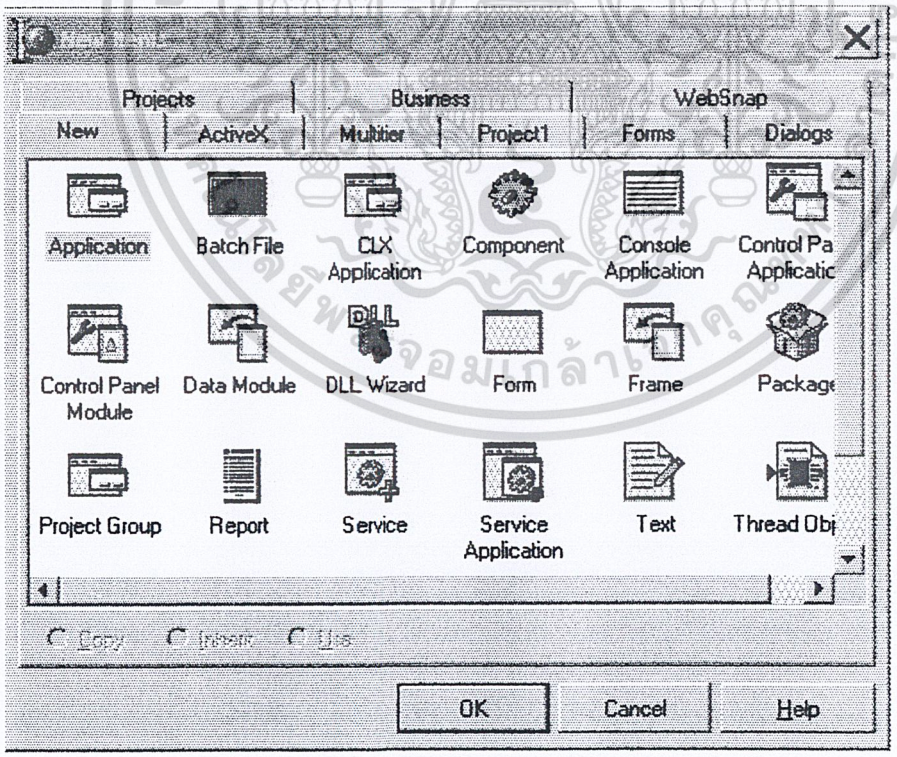
สามารถนำฟอร์มที่ถูกสร้างขึ้น มาใช้ในโปรเจกต์ได้ ซึ่งไฟล์ที่เก็บข้อมูลของฟอร์มจะอยู่ในรูปของ *.PAS สามารถเพิ่มฟอร์มได้โดยการ Click ที่เมนู Project>Ass to Project

การลบฟอร์มออกจากโปรเจกต์

ในกรณีที่สร้างฟอร์มขึ้นมาแล้ว แต่ไม่ต้องการใช้ฟอร์มนั้นอีกต่อไป สามารถลบฟอร์มนั้นออกจากโปรเจกต์ได้ โดยใช้คำสั่งเมนู Project>Remove from Project

การสร้างไอเทมอื่น ๆ (New)

นอกจากโปรเจกต์และฟอร์มปรกติแล้ว ยังมีไอเทมอีกหลายชนิดที่สร้างขึ้นได้ใน Delphi โดยเมื่อเลือกเมนู File>New จะปรากฏหน้าต่าง New Items สำหรับการสร้างไอเทมขึ้นดังรูป

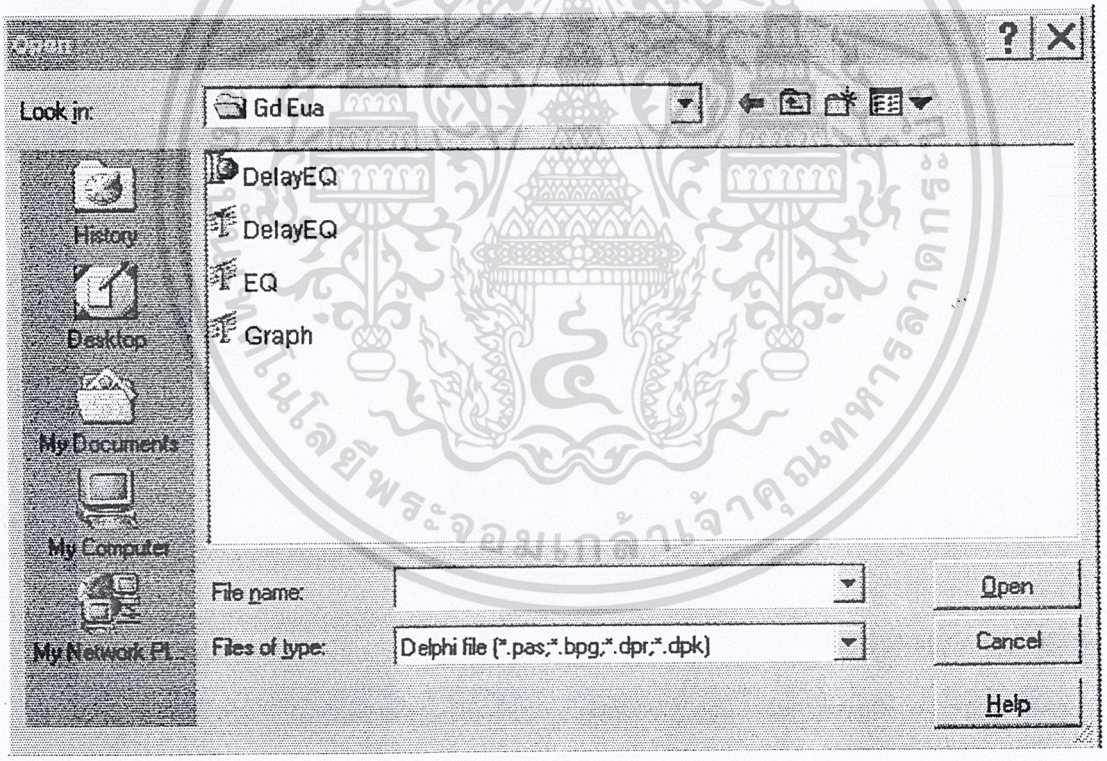


รูปที่ 3.11 แสดงการสร้างไอเทมอื่น ๆ

ในหน้าต่าง New Items จะประกอบด้วยต้นแบบ (Template) และวิซาร์ด (Wizard) ของฟอร์มและ Project ที่สามารถใช้เป็นตัวเริ่มต้นในการสร้างแอปพลิเคชั่นได้ ในหน้าต่าง New Items จะมีหลายหน้า แต่ละหน้าจะประกอบไปด้วยต้นแบบและวิซาร์ดลักษณะต่าง ๆ สามรถเลือกไอเทมเหล่านั้นมาวางลงบนฟอร์มโดยตรง หรือสามารถสร้างเป็น Inherit จากไอเทมนั้น (Inherit ก็คือการสร้างออปเจกใหม่ที่มีคุณสมบัติเริ่มต้น เช่นเดียวกับออปเจกเดิม โดยสามารถเพิ่มหรือเปลี่ยนแปลงคุณสมบัติต่าง ๆ เพื่อนำไปใช้ในลักษณะที่แตกต่างไป) การสร้างฟอร์มและ Project โดยใช้ต้นแบบและวิซาร์ดในวินโดว์ New Items นี้ จะช่วยเพิ่มความสะดวกและลดเวลาในการพัฒนาแอปพลิเคชั่นได้

การเปิดไฟล์ (Open)

สามารถเปิดไฟล์ที่ได้สร้างแล้วโดยเลือกเมนู File>Open เพื่อเปิดโปรเจกต์ (.DPR) หรือฟอร์ม (.PAS) ที่ได้มีการสร้างเอาไว้อยู่แล้วได้

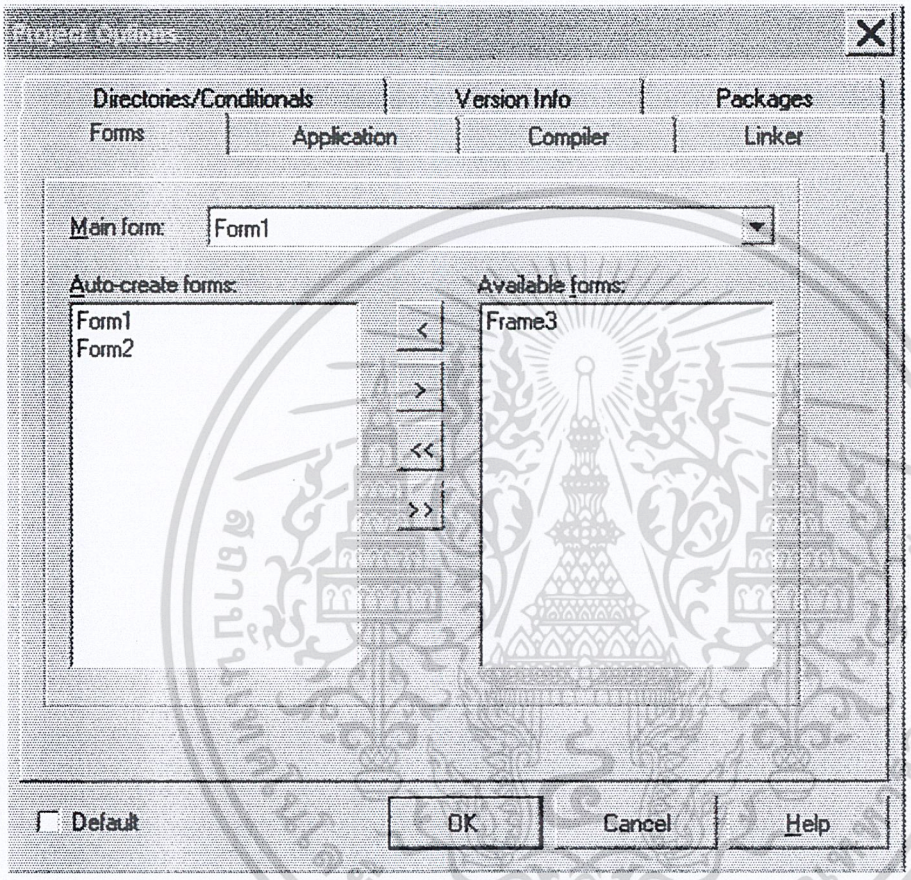


รูปที่ 3.12 แสดงการเปิดไฟล์ (Open)

การกำหนดลักษณะของโปรเจกต์

คุณสมบัติส่วนใหญ่ของโปรเจกต์ สามารถกำหนดได้โดยเลือกเมนู Project>Options ซึ่งจะปรากฏไดอะล็อก Project Options จากไดอะล็อกนี้สามารถกำหนดคุณสมบัติต่างๆ ของโปรเจกต์ เพื่อให้ได้แอปพลิเคชั่นที่ต้องการ ในไดอะล็อก Project Options จะประกอบไปด้วยหน้าหลาย ๆ แท็ป

1. Forms



รูปที่ 3.13 แสดงรูปลักษณะของ Form

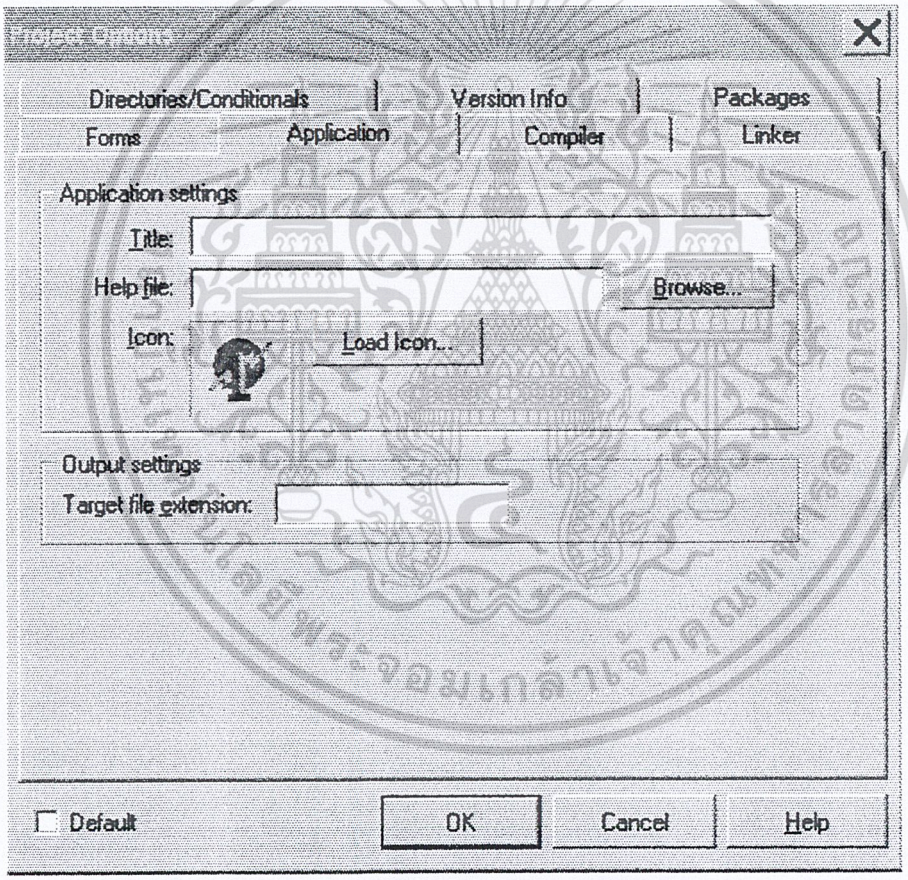
จะแสดงลิสต์ของชื่อฟอร์มที่มีใน Project สองลิสต์ คือ

- Auto-create forms จะเป็นลิสต์ของชื่อฟอร์มที่ Delphi จะทำการสร้างฟอร์มขึ้นเมื่อแอปพลิเคชั่นนั้นเริ่มต้นใช้งาน
- Available forms เป็นลิสต์ของชื่อฟอร์มที่มีอยู่ในโปรเจกต์ แต่ไม่ใช่ฟอร์มที่ถูกสร้างขึ้น เมื่อแอปพลิเคชั่นเริ่มต้นการใช้งาน โดยหากต้องการเรียกใช้งานฟอร์มเหล่านี้ จะต้องทำการเรียกฟอร์มเหล่านี้ขึ้นมาใช้งานเอง

สามารถย้ายชื่อฟอร์มระหว่างลิสต์ทั้งสองได้ โดย Click ที่ปุ่ม >> เพื่อย้าย Available forms ไปยัง Auto-create forms

ในการพัฒนาแอปพลิเคชั่นที่มีฟอร์มมาก ๆ ควรย้ายชื่อฟอร์มที่ไม่ใช่ฟอร์มที่ถูกเรียกใช้ตอนแอปพลิเคชั่นเริ่มทำงาน และฟอร์มที่ไม่ถูกเรียกใช้บ่อย ไปไว้ในลิสต์ Available forms ทั้งหมด เพื่อทำการสร้างฟอร์มเหล่านี้เมื่อมีการเรียกใช้งานเท่านั้น หากไม่ทำการย้ายฟอร์มเหล่านี้ไป เมื่อเปิดแอปพลิเคชั่นจะทำให้มีการสร้างฟอร์มจำนวนมาก ทำให้เสียเวลาในการเรียนแอปพลิเคชั่น และเปลืองหน่วยความจำ

2. Application



รูปที่ 3.14 แสดงลักษณะของ Application

ในหน้าแอปพลิเคชั่น สามารถกำหนดรายละเอียดต่าง ๆ ของแอปพลิเคชั่นที่จะสร้างขึ้นดังนี้

Settings	Descriptions
Application Settings	
Title	ข้อความที่จะแสดงเมื่อไอคอนถูกลดขนาด (Minimize) มีความยาวได้ไม่เกิน 255 ตัวอักษร
Help File	ชื่อของ Help ไฟล์ (.HLP) ของแอปพลิเคชั่น ไฟล์นี้จะถูกเรียกเมื่อมีการเรียกใช้ Help ภายในแอปพลิเคชั่น สามารถ Click ที่ปุ่ม Browse เพื่อแสดงไดอะล็อก Application Help File ในการเลือกชื่อไฟล์ที่ต้องการ
Icon	กำหนดไอคอน (.ICO) ที่ใช้เป็นไอคอนของแอปพลิเคชั่น ให้ Click ที่ปุ่ม Load Icon เพื่อเลือกไอคอนที่ต้องการ
Output Settings	
Target file extension	กำหนดนามสกุลของไฟล์ (File extension) ที่จะใช้หลังจากคอมไพล์แล้ว เช่น ใช้เป็น.OCX สำหรับ ActiveX แอปพลิเคชั่น หรือ .DLL สำหรับ DLL

ให้ Click ที่เช็คบ็อก Default หากต้องการให้ทุก ๆ โปรเจกต์ที่สร้างขึ้นใหม่ ใช้คุณสมบัติที่ได้กำหนดเอาไว้นี้

ไฟล์ประเภทต่าง ๆ ของ Delphi

เมื่อทำการบันทึกโปรเจกต์ในโฟลเดอร์ปลายทาง จะมีไฟล์ต่างๆ เกิดขึ้น การทำความรู้จักกับไฟล์เหล่านี้ จะช่วยให้สามารถจัดการกับไฟล์ต่างๆ ได้ดีขึ้น

ใน Delphi ประกอบด้วยไฟล์ต่างๆ ดังนี้

- Project File (.DPR)
- Unit File (.PAS, .DCU)
- Form File (.DFM)
- Resource File (.RES)

Project File (.DPR)

ทุกๆ โปรเจกต์ใน Delphi จะมีโค้ดภาษา Objet Pascal ที่ถูกคอมไฟล์เป็นแอปพลิเคชั่น โดยตัวกลางของแอปพลิเคชั่นเรียกว่า โปรเจกต์ไฟล์ ซึ่งไฟล์ตัวนี้ Delphi จะเป็นผู้เขียนขึ้นมาให้เป็นส่วนใหญ่

โปรเจกต์ไฟล์จะเป็นตัวอ้างอิงถึงทุก Unit และทุกฟอร์มที่มีภายในโปรเจกต์ คือเป็นตัวบอกว่าภายในโปรเจกต์มีไฟล์ อะไรบ้างที่ต้องนำมาใช้ ซึ่งปกติแล้วโปรเจกต์ไฟล์จะมีนามสกุล .DPR (Delphi Project)

นอกจากโปรเจกต์ไฟล์ ยังมีอีกหนึ่งไฟล์ที่ใช้สำหรับเก็บค่าตัวเลือกต่างๆ ที่กำหนดเอาไว้ใน Project Options (เมนู Project>Options) ซึ่งเป็นไฟล์ที่มีชื่อเดียวกันกับโปรเจกต์ไฟล์ แต่มีนามสกุลเป็น .DOF (Delphi Option File)

โดยปกติ ไม่ควรทำการแก้โค้ดที่อยู่ในโปรเจกต์ไฟล์ ตัว Delphi จะทำการแก้ไขปรับปรุงไฟล์นี้ให้ถูกต้องเอง การแก้ไขสิ่งต่างๆ ภายในโปรเจกต์สามารถทำได้ผ่าน Project Manager (เรียกจากเมนู View>Project Manager)

การแสดงโค้ดของโปรเจกต์ไฟล์ ให้เรียก Project Manager โดยเลือกเมนู View>Project Manager แล้ว Click Mouse ปุ่มขวาบนชื่อโปรเจกต์ แล้วเลือก View Source

Unit File (.PAS, .DCU)

Object Pascal สามารถแยกโค้ดออกเป็นหลายๆ ส่วนได้ โดยแต่ละส่วนเรียกว่า Unit ซึ่ง Unit ส่วนใหญ่จะเป็น Unit สำหรับฟอร์ม ซึ่งเป็น Unit สำหรับการเขียนโปรแกรมเพื่อควบคุมฟอร์ม แต่ก็สามารถสร้าง Unit ที่ไม่อ้างอิงถึงฟอร์มได้

ใน Unit ต่างๆ เมื่อทำการ Save จะได้เป็นไฟล์ .PAS (Pascal)

เมื่อทำการคอมไพล์ Unit เหล่านี้ Delphi จะสร้างไฟล์ .DCU (Delphi Compiled Unit) ซึ่งเป็นไบนารีไฟล์ชื่อเดียวกันกับ Unit

Unit ไฟล์สำหรับฟอร์ม

เมื่อมีการสร้างฟอร์มใหม่ จะมีการสร้าง Unit ไฟล์สำหรับฟอร์มนั้น ซึ่งจะมีโค้ดบางส่วนที่ระบุถึงคุณสมบัติต่างๆ ของฟอร์มไว้ให้อยู่แล้ว ลักษณะของ Unit ไฟล์สำหรับฟอร์มจะมีหน้าตาดังนี้

ภายใน type declaration จะเป็นการกำหนดออปเจกต์ของฟอร์ม ซึ่งสืบทอดมาจากคลาส Tform และใน Private declarations (ใต้คำว่า private) และใน Public declarations (ใต้คำว่า public) สามารถกำหนด ตัวแปร Function และ Procedure สำหรับออปเจกต์ของฟอร์มใหม่ขึ้นเองได้ตามลำดับ (โดยในแบบ Private จะอ้างอิงถึงได้เฉพาะใน Unit นี้เท่านั้น และ แบบ Public จะอ้างอิงถึงได้จาก Unit อื่น ๆ)

ใต้ implementation จะมี $R compiler directive {$R*.DFM} ซึ่งเป็นตัวเชื่อมต่อไปยังฟอร์มไฟล์ ทำให้มีการเรียกใช้ฟอร์มไฟล์ในโปรเจกต์ เมื่อทำการคอมไพล์เป็นไฟล์สำหรับเรียกใช้งาน

From File (.DFM)

เมื่อมีการสร้างฟอร์มใหม่ Delphi จะสร้างฟอร์มไฟล์หนึ่งไฟล์ และ Unit ไฟล์หนึ่ง ไฟล์ฟอร์มไฟล์จะเป็นที่เก็บรายละเอียดเกี่ยวกับฟอร์มที่เราสร้างขึ้น สามารถเปิดฟอร์มไฟล์ใน Code Editor เพื่อทำการแก้ไขรายละเอียดของฟอร์มได้ โดย Click Mouse ปุ่มขวาที่ฟอร์มแล้ว เลือก View as Text หรือกลับไปแก้ไขแบบเป็นฟอร์มเหมือนเดิมโดย Click Mouse ปุ่มขวาที่ชื่อ ของฟอร์มไฟล์บน Code Editor แล้วเลือก View as Form

ชื่อของฟอร์มไฟล์จะเป็นชื่อเดียวกับ Unit ไฟล์ที่อ้างอิงถึงมัน แต่จะมีนามสกุลที่แตกต่าง กัน โดย Unit ไฟล์จะมีนามสกุลเป็น .PAS ส่วนฟอร์มไฟล์จะมีนามสกุลเป็น .DFM (Delphi Form)

โดยปกติ เมื่อเรา Save ไฟล์ต่างๆ ข้างต้น Delphi จะทำการสร้างไฟล์สำรองให้ โดย เปลี่ยนนามสกุลเป็น .~DP, .~PA และ .~DF สำหรับโปรเจกไฟล์ (.DPR), Unit ไฟล์ (.PAS) และฟอร์มไฟล์ (.DFM) ตามลำดับ

Resource File (.RES)

เมื่อทำการคอมไพล์โปรเจกต์ ตัวโปรแกรม Delphi จะทำการสร้าง Resource ไฟล์ซึ่งมีชื่อ เดียวกันกับโปรเจกต์ไฟล์ แต่มีนามสกุลเป็น .RES Resource ไฟล์ที่ Delphi สร้างขึ้นมาให้นี้ เป็น ไฟล์ตามมาตรฐานของ Window

Resource ไฟล์ เป็นไฟล์สำหรับเก็บไอเทมต่างๆ เช่น ไอคอนของแอปพลิเคชั่น สามารถ สร้างหรือแก้ไข Resource ไฟล์ โดยการเพิ่ม Resource อื่นๆ เข้าไปได้เช่น รูปภาพ (Bitmap), เคอเซอร์, ไอคอน หรือข้อความ

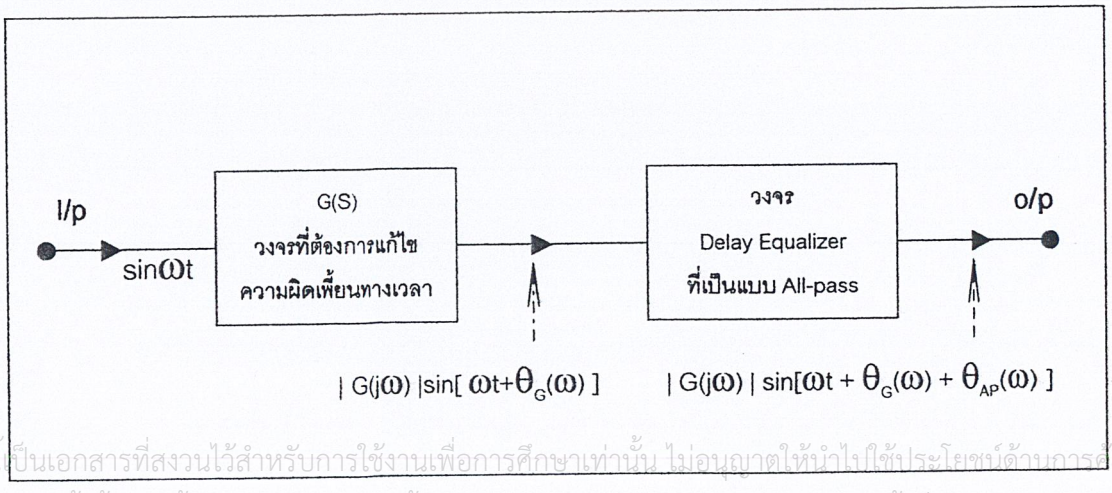
บทที่ 4
การออกแบบ

4.1 การออกแบบวงจรดีเลย์อิควอไลเซอร์ (Delay Equalizer)

บทนี้ได้บรรยายถึงความสำเร็จของวิธีการปรับค่าผิดเพี้ยนกรุ๊ฟดีเลย์โดยใช้วิธีการกำหนดค่าประมาณเริ่มต้นขึ้นมาก่อน และประยุกต์ใช้วิธีนิวตัน-ราฟสันเพื่อปรับค่าประมาณให้ได้ผลตามต้องการ กระบวนการนี้จะทำซ้ำๆ จนกระทั่งได้ค่าเป็นที่น่าพอใจ

ปัญหาของการปรับกรุ๊ฟดีเลย์นั้น ได้มีความพยายามที่จะใช้เทคนิคมากมาย วัตถุประสงค์ก็เพื่อให้ได้ค่าตามต้องการ ค่าerror สูงสุดต้องน้อยกว่าค่าความคลาดเคลื่อน(tolerance)ที่กำหนดไว้ กระบวนการที่ยุ่งยากซับซ้อนนี้สามารถประสบความสำเร็จได้โดยใช้คอมพิวเตอร์ในการคำนวณ ในการออกแบบการติดต่อสื่อสารโทรคมนาคม เช่น ระบบรับส่งของสัญญาณโทรทัศน์และพัลซ์ทรานส์มิสชั่น (Pulse Transmission) เรามีความจำเป็นที่จะต้องชดเชยทั้งทางเฟสและทางขนาดของสัญญาณ โดยทำให้เฟสที่ได้เป็นเชิงเส้นมากที่สุดตามที่ต้องการในช่วง ทรานส์มิสชั่นแบนด์ (Transmission Band) หรืออีกนัยหนึ่งก็คือทำให้กรุ๊ฟดีเลย์ (Group Delay) คงที่ บทนี้จะเสนอหลักในการประมาณฟังก์ชั่นของวงจรดีเลย์อิควอไลเซอร์ เพื่อแก้ไขความผิดเพี้ยนทางเวลาเพื่อให้ได้กรุ๊ฟดีเลย์คงที่ โดยใช้คอมพิวเตอร์ช่วยอิเตอเรท (Iterate) ในการหาคำตอบ ซึ่งเน็ทเวิกร์ฟังก์ชั่นที่ได้จะต้องเป็นวงจรที่ยอมให้สัญญาณผ่านได้หมดทุกความถี่ (All-pass Network) และทรานส์เฟอร์โคเอฟฟิกเซี่ยนของมันเป็นฟังก์ชั่นที่มีซีโร่อยู่ทางซีกขวา และมีโพลอยู่ทางซีกซ้ายของแกนอิมเมจินารี่ (Imaginary Axis) ใน S-Plane

การแก้ไขความผิดเพี้ยนทางเวลากระทำได้โดยการนำเอาวงจรดีเลย์อิควอไลเซอร์แบบ All-pass ที่สร้างขึ้นมาต่อคาสเคดเข้ากับวงจรที่ต้องการแก้ไขความผิดเพี้ยนทางเวลาตามในรูปที่ 4.1



รูปที่ 4.1 แสดงการนำดีเลย์อิควอไลเซอร์ไปใช้งานร่วมกับวงจร

การออกแบบดีเลย์อิควอไลเซอร์กำหนดให้ทรานส์เฟอร์ฟังก์ชั่น

$$H(s) = \frac{P(-s)}{P(s)} \tag{4.1}$$

โดยที่ P(s) เป็นเฮอร์วิทซ์โพลิโนเมียล

เมื่อเขียนให้อยู่ในแกนของความถี่ (Real Frequency Axis) จะได้

$$H(j\omega) = Me^{j\theta} = \frac{P(-j\omega)}{P(j\omega)} \tag{4.2}$$

เมื่อ M และ θ เป็นขนาดและเฟสตามลำดับ

โดยที่

$$|P(j\omega)| = |P(-j\omega)| \tag{4.3}$$

และ

$$-\arg P(j\omega) = \arg P(-j\omega) \tag{4.4}$$

จากสมการ (4.2) จะได้ M = 1 และ $\theta = -2 \arg P(j\omega)$ และเมื่อเขียนเป็นทรานส์เฟอร์ฟังก์ชั่นอันดับ (order) ที่ 2 จะได้

$$H(s) = \frac{s^2 - as + b}{s^2 + as + b} \tag{4.5}$$

ทรานส์เฟอร์ฟักชั่น H(s) ในสมการ (4.5) ซึ่งอยู่ในรูปความถี่เชิงซ้อน (Complex Frequency) สามารถเขียนตำแหน่งของโพลและซีโร่ใน S-Plane แสดงได้ดังรูปที่ 4.1 จากทราส์เฟอร์ฟังก์ชั่นในสมการ (4.5)สามรถเขียนได้ใหม่เป็น

$$H(s) = \frac{s^2 - 2ps + p^2 + q^2}{s^2 + 2ps + p^2 + q^2} \tag{4.6}$$

ค่าของเฟสในสมการ 4.6 จะได้

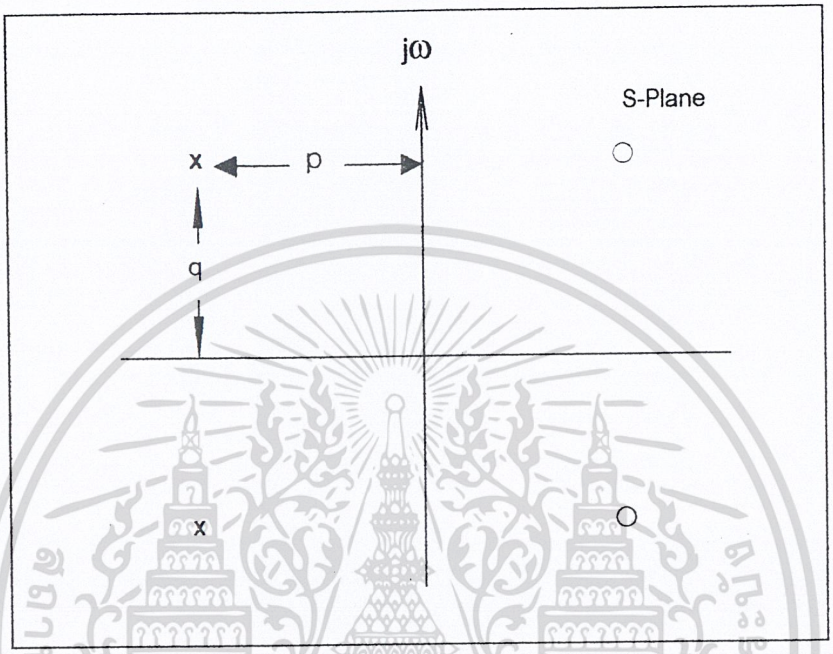
$$\theta = -2\tan^{-1}\frac{\omega - q}{p} - 2\tan^{-1}\frac{\omega + q}{p} \tag{4.7}$$

จากสมการ (4.7)ค่ากรุ๊ฟดีเลย์สามารถเขียนได้คือ

$$T_d = \frac{2p}{p^2 + (\omega - q)^2} + \frac{2p}{p^2 + (\omega + q)^2} \tag{4.8}$$

จากสมการ (4.8) จะเห็นได้ว่าค่ากรุ๊ฟดีเลย์ที่ได้ขึ้นอยู่กับระยะห่างของโพลหรือซีโร่ S-Plane คือตัวแปร p และ q ฉะนั้นการประมาณเน็ทเวิร์กฟังก์ชั่นเพื่อแก้ไขความผิดเพี้ยนทางเวลาของวงจรที่ต้องการนั้นก็คือการบังคับหาตำแหน่งของโพลหรือซีโร่ที่ดีที่สุดในวงจรดีเลย์อิควอไลเวอร์ เพื่อให้

ได้ผลรวมของกรุ๊ฟดีเลย์ที่ได้คงที่ ซึ่งจำนวนของโพลหรือซีโร่ (อันดับของทรานส์เฟอร์ฟังก์ชั่น) ขึ้นอยู่กับช่วงแบนด์วิธของความถี่ที่ต้องการและขนาดค่าคลาดเคลื่อนสูงสุดที่ยอมให้ได้ในกรุ๊ฟดีเลย์ช่วงนั้น



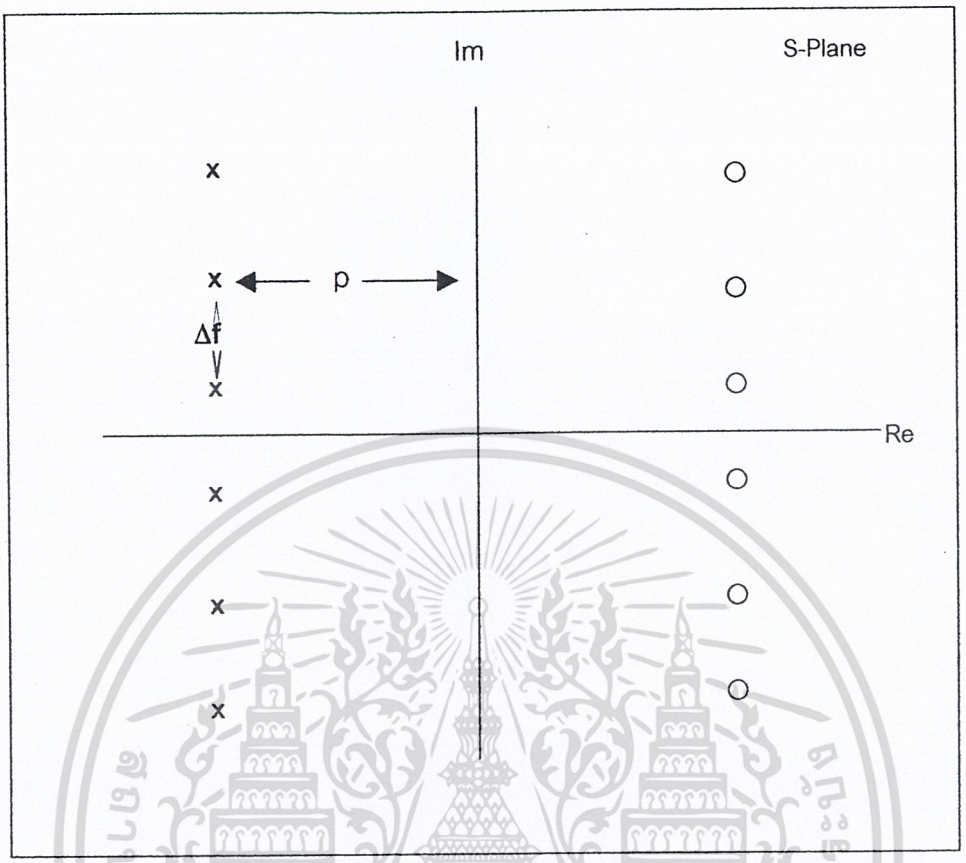
รูปที่ 4.2 แสดงตำแหน่งโพลและซีโร่ของ All-pass Tranfer Function อันดับที่2

ในการประมาณค่าเริ่มต้นของโพลและซีโร่ในทรานส์เฟอร์ฟังก์ชั่นของ All-pass นั้นจะใช้วิธีการกำหนดค่าของโพลและซีโร่กระจายขนานไปกับแกนจินตภาพ (Imaginary) ใน S-Plane ดังแสดงในรูปที่ 4.3 ช่วงระยะห่างระหว่างโพลหรือซีโร่นั้น สามารถกำหนดได้จากช่วงแบนด์วิธของความถี่ที่ต้องการกับจำนวนเซคชั่นของ All-Pass อันดับที่ 2 คือ

$$\Delta f = \frac{\textit{แบนด์วิธของความถี่}}{\textit{จำนวนเซคชัน} - 0.5} \qquad (4.9)$$

จากรูปที่ 4.3 เราจะได้ผลรวมของกรุ๊ฟดีเลย์คือ

$$T_d = \frac{1}{\pi}\sum_{k=-\infty}^{\infty}\frac{p}{p^2 + \Delta f^2 (f-k)^2}$$



รูปที่ 4.3 แสดงการกำหนดค่าเริ่มต้นของโพลและซีโร่

โดยจะปรากฏค่าจุดสูงสุดที่ f=±(k-1/2) เพราะฉะนั้นเมื่อกำหนดให้ i = 2k จะได้ค่ากรุ๊ฟดีเลย์ที่จุดสูงสุดและค่าต่ำสุดคือ

$$T_{d_{\min}} = \frac{2}{\pi\Delta f}\sum_{l=0,-2,\ldots}^{2,4,\ldots}\frac{2p/\Delta f}{(2p/\Delta f)^2+i^2} = \frac{1}{\Delta f}\coth\frac{p\pi}{\Delta f}$$

$$T_{d_{\max}} = \frac{2}{\pi\Delta f}\sum_{i=-1,-3,\ldots}^{1,3,\ldots}\frac{2p/\Delta f}{(2p/\Delta f)^2+i^2} = \frac{1}{\Delta f}\tanh\frac{p\pi}{\Delta f}$$

ค่าเฉลี่ยของกรุ๊ฟดีเลย์จะได้

$$T_{d_0} = \frac{1}{2\Delta f}\left(\coth\frac{p\pi}{\Delta f}+\tanh\frac{p\pi}{\Delta f}\right) = \frac{1}{\Delta f}\coth\frac{2p\pi}{\Delta f}$$

ซึ่งจะได้ขนาดค่าคลาดเคลื่อนสูงสุดของกรุ๊ฟดีเลย์คือ

$$\in = \frac{2}{\Delta f}\left(\coth\frac{p\pi}{\Delta f}-1\right) \cong \frac{4}{\Delta f}e^{-2p\pi/\Delta f}$$

ฉะนั้นจะได้ระยะห่างของโพลหรือซีโร่กับแกน Imaginary คือ

$$p = \frac{\Delta f}{2\pi}\ln\frac{4}{\Delta f \in} \qquad (4.10)$$

และจะได้ระยะห่างของโพลหรือซีโร่กับแกน Real คือ

$$q_i = \Delta f\left(i - \frac{1}{2}\right) \qquad (4.11)$$

โดยที่ i เป็นลำดับที่ของวงจร All-pass แต่ละชุด (section)

ตำแหน่งของโพลหรือซีโร่ในเน็ทเวิกร์ฟังก์ชั่นอันดับที่ 2 ที่ประมาณขึ้นนั้น จะมีผลต่อกรุ๊ฟดีเลย์ คือ ถ้า p มีค่าน้อยจะทำให้ยอดของกรุ๊ฟดีเลย์มีค่าสูงและมีแบนด์วิธที่แคบ และ ถ้า p มีค่ามากจะทำให้ยอดของกรุ๊ฟดีเลย์มีค่าต่ำและมีแบนด์วิธที่กว้าง ส่วนตัวแปร q จะมีผลต่อแกนทางความถี่คือจะเป็นตำแหน่งของความถี่ ณ.ที่จุดยอดของกรุ๊ฟดีเลย์นั้น จากค่าประมาณที่ได้จะเป็นค่าเริ่มต้นเพื่อให้คอมพิวเตอร์ช่วยอินเตอเรทในการหาคำตอบต่อไป ซึ่งถ้าค่าเริ่มต้นที่ให้มีค่าเริ่มจากการกำหนดให้ D เป็นผลรวมของกรุ๊ฟดีเลย์ทั้งหมด คือ ของวงจรดีเลย์อิควอไลเซอร์ และวงจรที่ต้องการแก้ไขความผิดเพี้ยนทางเวลา และมีค่าเฉลี่ยคงที่เป็น D_0 ด้วยจุดสูงสุดและต่ำสุดเปลี่ยนแปลงจากค่านั้นเท่ากับ $\pm\epsilon/2$ ฉะนั้นที่ความถี่ใด ๆ สมมติให้เป็น j จะได้

ค่าต่ำสุด: $D_j + \epsilon/2 = D_0$

ค่าสูงสุด: $D_j - \epsilon/2 = D_0$ (4.12)

ค่าผลรวามของกรุ๊ฟดีเลย์ที่กำหนดจากค่าเริ่มต้นนั้นจะมีความคลาดเคลื่อน สมมติให้เป็น ΔD ฉะนั้นจากสมการที่ (4.12) เป็น

ค่าต่ำสุด: $\Delta D_j + D_j + \epsilon/2 = D_0$

ค่าสูงสุด: $\Delta D_j + D_j - \epsilon/2 = D_0$

(4.13)

ผลรวมของกรุ๊ฟดีเลย์ทั้งหมด D จะขึ้นอยู่กับโคออดิเนทของ (p,q) ในแต่ละส่วนของทรานส์เฟอร์ฟังก์ชั่น All-pass อันดับที่สอง การเปลี่ยนแปลงค่าใด ๆ ของ D ที่ช่วงความถี่ต่าง ๆ ขึ้นอยู่กับการเปลี่ยนค่า $(\Delta p, \Delta q)$ ในโคออดิเนทที่เหมาะสม

ตัวอย่างเมื่อสมมติให้ (p_0, q_0) เป็นรากเริ่มต้นของระบบสมการ

$$\Delta D(p, q) = 0$$

ถ้า $(p_0+\Delta p, q_0+\Delta q)$ เป็นรากของระบบสมการแล้วจะได้

$$\Delta D(p_0 + \Delta p, q_0 + \Delta q) = 0$$

ฟังก์ชัน ΔD สามารถหาอนุพันธ์ได้ เมื่อกระจายออกเป็นอนุกรมโดยใช้อนุกรมเทย์เลอร์หลายตัวแปรจะได้

$$\Delta D_0 + \Delta p \frac{\partial}{\partial p_0} \Delta D + \Delta q \frac{\partial}{\partial q_0} \Delta D + \ldots = 0$$

เมื่อไม่คิดเทอมที่มีอนุพันธ์อันดับสองและสูงกว่าแล้วจะได้ระบบสมการเชิงเส้น

$$\Delta p \frac{\partial}{\partial p_0} \Delta D + \Delta q \frac{\partial}{\partial q_0} \Delta D = -\Delta D_0$$

ดังนั้นรากที่ประมาณได้ใหม่คือ

$$p_1 = p_0 + \Delta p$$
$$q_1 = q_0 + \Delta q$$

ถ้าต้องการหารากที่ดียิ่งขึ้น ต้องดำเนินการกระทำซ้ำหลายหนจนกระทั่งถูกต้องตามที่ต้องการ

จากหลักการดังกล่าวเราสามารถนำเอาวิธีนิวตัน-ราฟสันมาใช้แก้สมการเชิงโค้งทางพีชคณิต 2 n ตัวแปรและ m สมการของวงจรดีเลย์อิควอไลเซอร์ที่ความถี่ใด ๆ จะได้ผลต่างของกรุ๊ฟดีเลย์ ΔD_j และนำไปหาความสัมพันธ์เพื่อหาการเปลี่ยนแปลงของ $(\Delta p_j, \Delta q_j)$ ในโคออดิเนท ซึ่งจะมีคุณสมบัติของการลู่เข้าที่ดีหลักการเบื้องต้นของวิธีนิวตันอิเตอเรชั่นก็คือใช้เทย์เลอร์กระจายแต่ละสมการออกเป็นอนุกรม เมื่อตัดเทอมที่มีอันดับสูงกว่าหนึ่งออกแล้วจะกลายเป็นการหารากของระบบสมการเชิงเส้น ซึ่งสามารถเขียนเป็นสมการได้คือ

$$\Delta D_j = \left(\frac{\partial D}{\partial p_1}\right)_{f_j} \Delta p_1 + \ldots + \left(\frac{\partial D}{\partial p_1}\right)_{f_j} \Delta p_i + \ldots +$$
$$+ \left(\frac{\partial D}{\partial p_N}\right)_{f_j} \Delta p_N + \ldots + \left(\frac{\partial D}{\partial q_i}\right)_{f_j} \Delta q_i + \ldots +$$
$$+ \left(\frac{\partial D}{\partial q_1}\right)_{f_j} \Delta q_1 + \ldots + \left(\frac{\partial D}{\partial q_N}\right)_{f_j} \Delta q_N \qquad (4.14)$$

เมื่อ $\left(\frac{\partial D_j}{\partial p_i}\right)$ และ $\left(\frac{\partial D}{\partial q_s}\right)$ เป็นการหาอนุพันธ์ของ D เทียบกับ p และ q ที่ความถี่ j ซึ่งสามารถหาได้คือ

$$\frac{\partial}{\partial p_i} D_j = 2\left[\frac{(\omega_j + q_j)^2 - p_i^2)}{\left[p_i^2 + (\omega_j + q_i)^2\right]^2} + \frac{(\omega_j - q_i)^2 - p_i^2)}{\left[p_i^2 + (\omega_j - q_i)^2\right]^2}\right] \qquad (4.15)$$

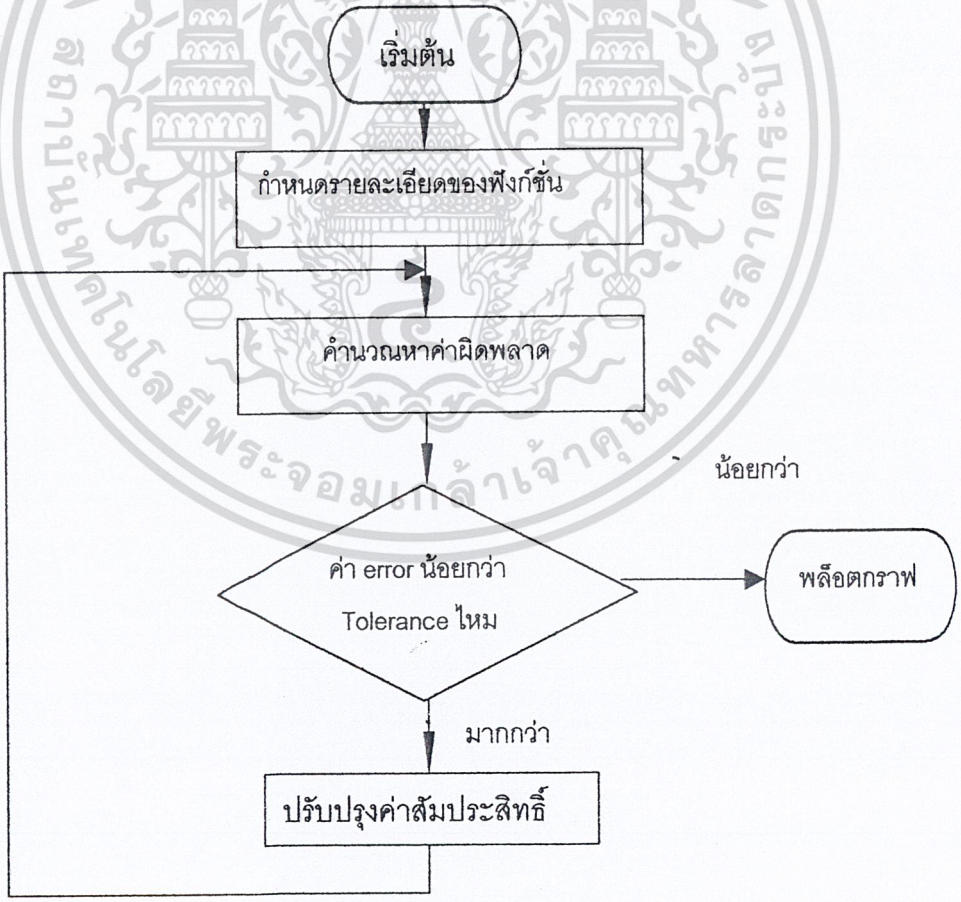
$$\frac{\partial}{\partial q_i} D_j = 4p_i\left[\frac{\omega_j + q_j}{\left[p_i^2 + (\omega_j + q_i)^2\right]^2} + \frac{\omega_j - q_i}{\left[p_i^2 + (\omega_j - q_i)^2\right]^2}\right] \qquad (4.16)$$

สมการที่ 4.14 เป็นสมการเชิงเส้นที่จะหาค่าของ Δp_j และ Δq_j เพื่อปรับปรุงค่าของ p_j และ q_j ใหม่แล้วนำไปแทนค่าเพื่อหาค่าผลรวมของกรุ๊ฟดีเลย์และจะได้ค่า ΔD_j น้อยลงเรื่อย ๆ โดยจะกระทำซ้ำหลายหนจนกระทั่ง ΔD_j มีค่าเป็นศูนย์ ในทางปฏิบัติกระบวนการจะกระทำจนกว่าจะได้ค่าต่ำสุด: $\left| D_j + \in /2 - D_0 \right| \le Tol$

ค่าสูงสุด: $$\left| D_j - \in /2 - D_0 \right| \le Tol \qquad (4.17)$$

เมื่อ Tol เป็นค่าตรวจสอบการลู่เข้าที่ยอมให้ได้ (Tolerance)

จากเซตของสมการที่กล่าวมาจะเห็นได้ว่ามีจำนวนของตัวแปรทั้งหมดเท่ากับ 2n+3 เมื่อ n เป็นจำนวนเน็ทเวิกร์ของวงจร All-pass อันดับที่สองที่มีตัวแปรคือ p_i และ q_i และตัวแปรที่เหลือคือ $D_0, \in$ และช่วงแบนด์วิธของความถี่ที่ใช้ ถ้าวงจรดีเลย์อิควอไลเซอร์ที่ออกแบบมีช่วงแบนด์วิธของความถี่ที่กว้างและต้องการให้มีค่าคลาดเคลื่อน ($\in$) ที่น้อย ก็จะต้องใช้จำนวนอันดับ (order) ของเน็ทเวิกร์ที่สูง ส่วนค่าเฉลี่ยคงที่ของกรุ๊ฟดีเลย์ นั้นถ้ามีค่าน้อยก็จะทำให้ช่วงแบนด์วิธของความถี่ที่ได้กว้างขึ้น และอาจทำให้มีค่าคลาดเคลื่อนที่มากขึ้นด้วย แต่ขึ้นอยู่กับวงจรที่เราต้องการแก้ไขความผิดเพี้ยนทางเวลาว่ามีความผิดเพี้ยนมากน้อยเพียงใด ซึ่งทั้งหมดจะมีความสัมพันธ์ต่อกันและสามารถเขียนเป็นโปรแกรมเพื่อหาค่าตัวแปรหรือสัมประสิทธิ์ของเน็ทเวิกร์ฟังชั่นของวงจรดีเลย์อิควอไลเซอร์ได้ดังแสดงเป็นโฟลว์ชาร์ทในรูปที่4.4 และมีรายละเอียดของโปรแกรมแสดงไว้ในภาคผนวก



รูปที่ 4.4 โฟลว์ชาร์ทแสดงการหาสัมประสิทธิ์ของเน็ทเวิกร์ฟังก์ชั่นของกรุ๊ฟดีเลย์อิควอไลเซอร์

บทที่ 5
ผลการทดลอง

ใช้วงจร All-pass อันดับ2 จำนวน 5 เซคชั่น ในช่วงแบนด์วิธ 6 MHz จะได้

$$\Delta f = \frac{6}{5-0.5} = 1.333 \tag{5.1}$$

หาตำแหน่งโพล และ ซีโร่ โดยกำหนดให้ค่า error น้อยกว่า 5 nS จะได้

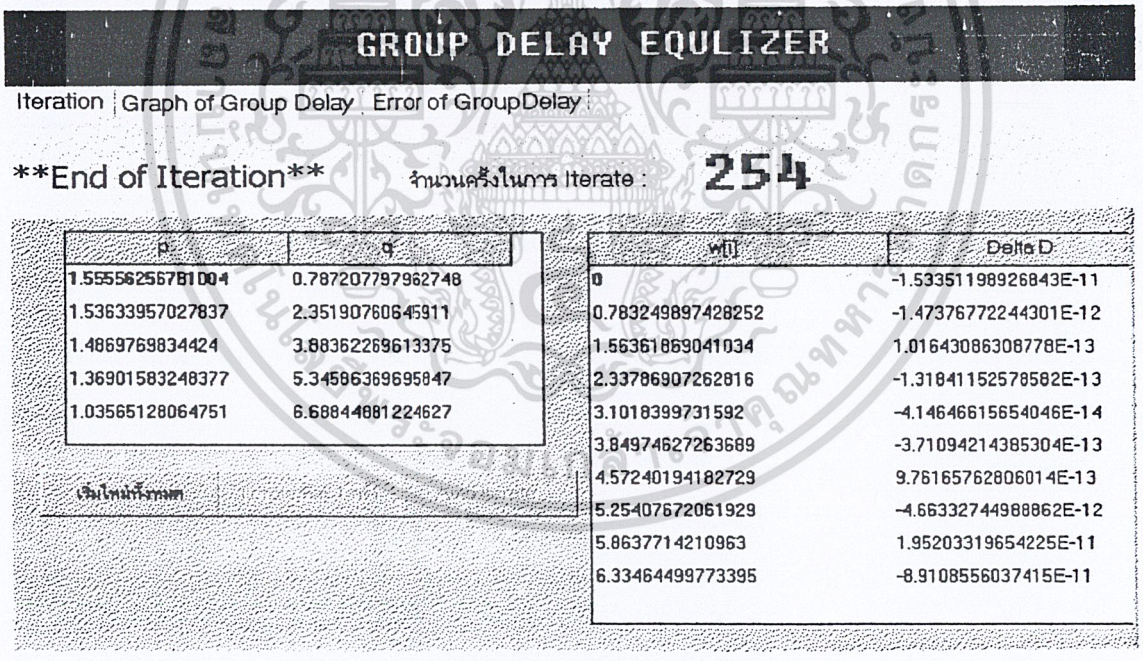
$$p = \frac{1.333}{2\pi} \ln \frac{4}{1.333 \times 0.005} = 1.3585 \tag{5.2}$$

และ

$$q = 1.333\left(i - \frac{1}{2}\right) = 0.667\ ,2\ ,3.333\ ,4.667\ ,6 \tag{5.3}$$

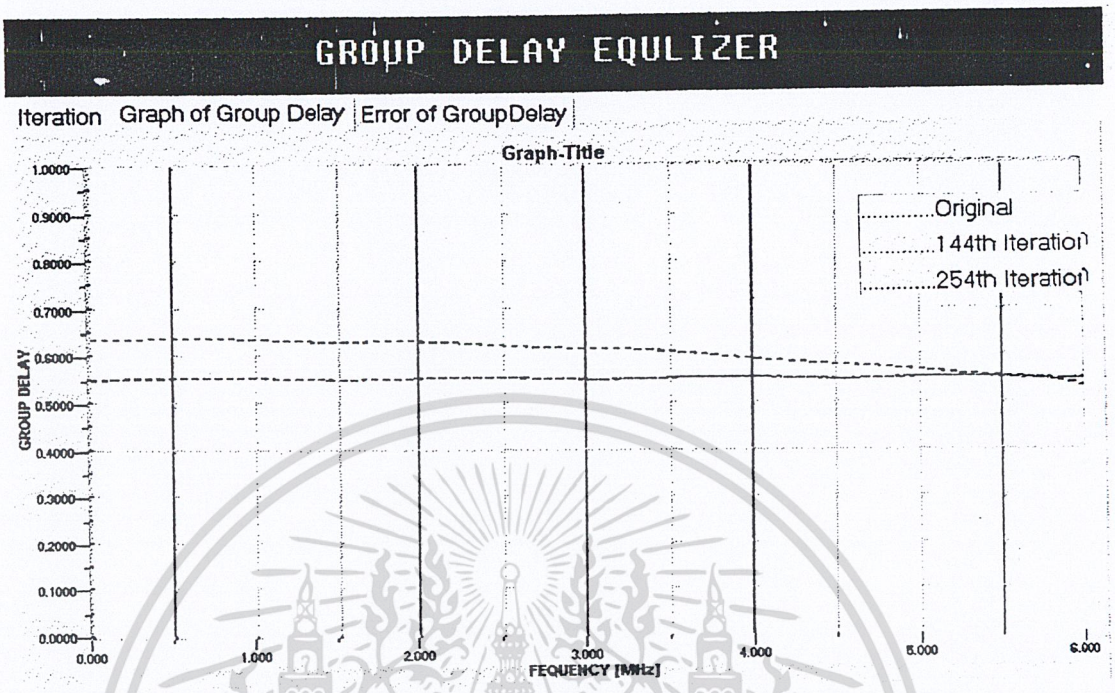
5.1การทดสอกร๊ฟดีเลย์ของ วงจรออพาส

ทำการอิเตอเรทโปรแกรม ของวงจร Delay Equalizer ที่เป็นแบบ All-pass เมื่อยังไม่มีวงจรที่ต้องการแก้ไข จะต้องอิเตอร์เรทั้งหมด 254 ครั้ง ได้ค่าต่างๆตามรูปที่ 5.1



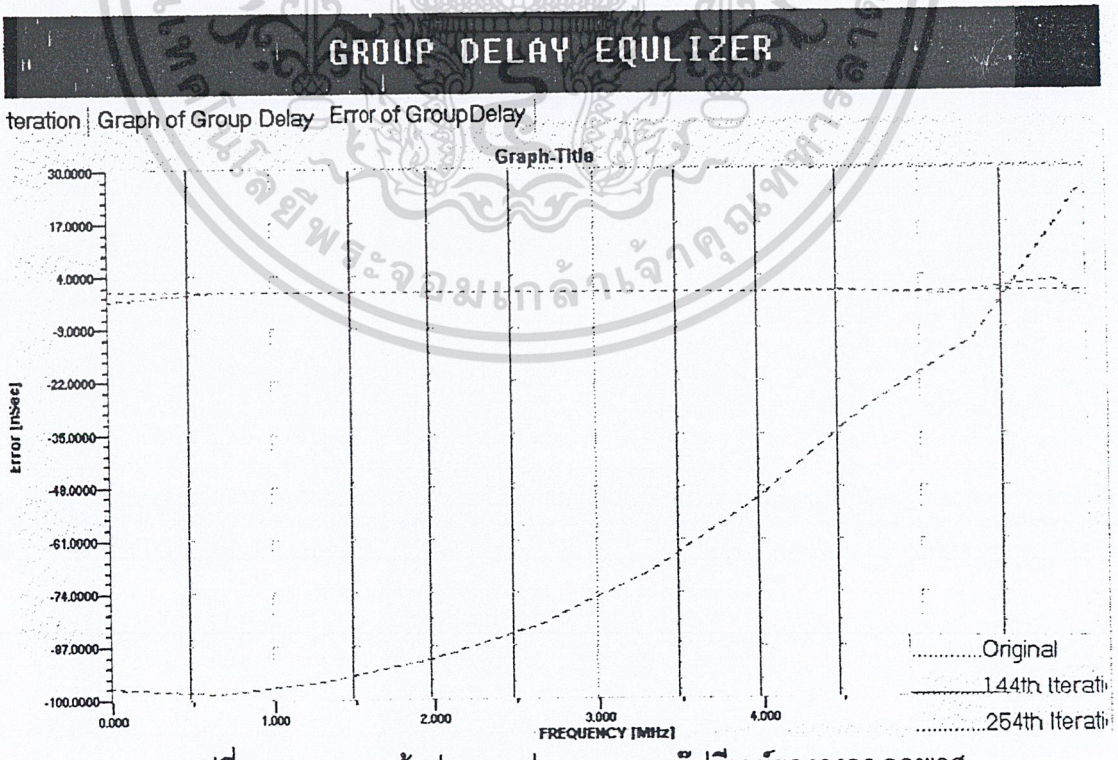
รูปที่ 5.1 แสดงหน้าต่างการอิเตอเรทของวงจร ออพาส

และจะได้กราฟของกรุ๊ฟดีเลย์ ดังรูปที่ 5.2



รูปที่ 5.2 แสดงหน้าต่างกราฟของกรุ๊ฟดีเลย์ของวงจร ออพาส

จะได้กราฟ error ของ กรุ๊ฟดีเลย์ ดังรูปที่ 5.3



รูปที่ 5.3 แสดงหน้าต่างกราฟ error ของกรุ๊ฟดีเลย์ของวงจร ออพาส

การทดสอบกรุ๊ฟดีเลย์ของวงจร โลพาส

ทำการปรับ group delay ของวงจร low pass filter ซึ่งมี ทรานส์เฟอร์ฟังค์ชั่น คือ

$$H(s) = \left[\frac{1}{s^4 + 2.7093s^3 + 3.8491s^2 + 2.6854s + 1}\right] \quad (5.4)$$

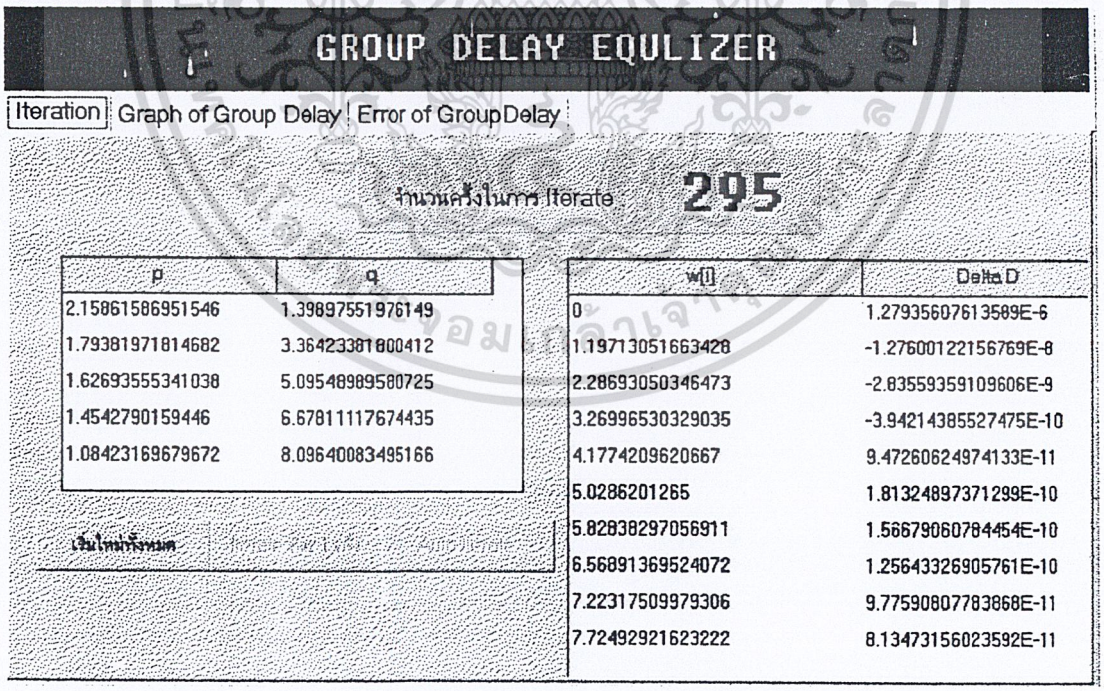
ค่าของเฟสในสมการ ที่ (5.4) คือ

$$\theta = \tan^{-1}\left[\frac{(2.7093\omega^3 - 2.6854\omega)}{(\omega^4 - 3.8419\omega^2 + 1)}\right] \quad (5.5)$$

จากสมการที่ (5.5) จะได้ สมการกรุ๊ฟดีเลย์ คือ

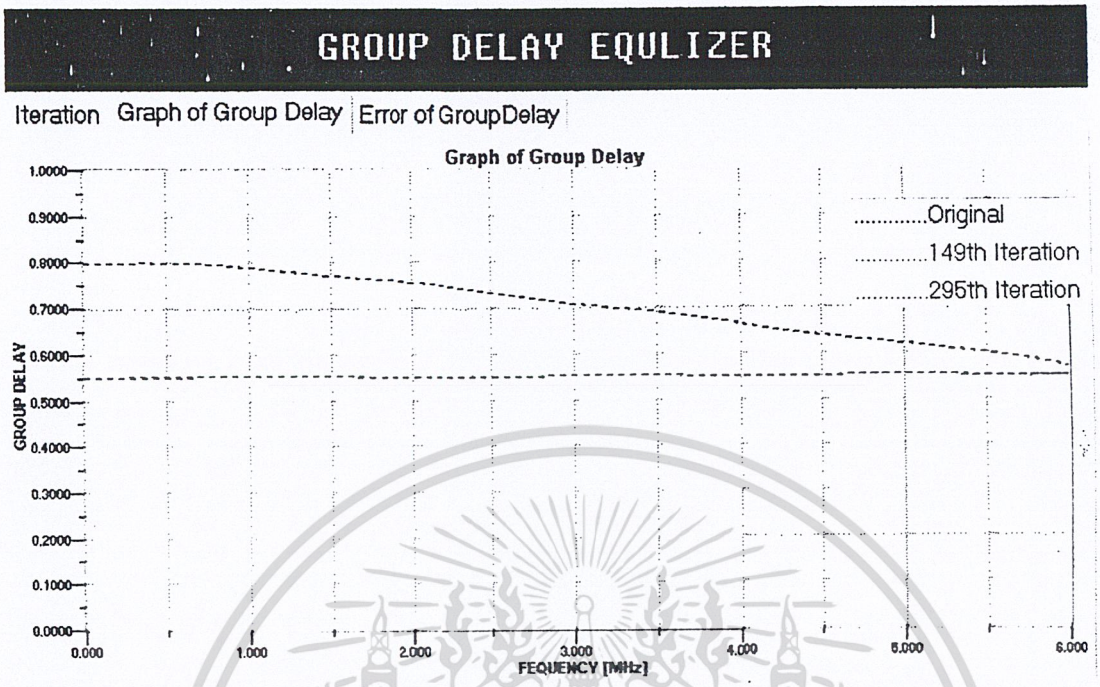
$$t_d = \left[\frac{(3.8491\omega^2 - \omega^4 - 1)}{(\omega^4 - 3.8491\omega^2 + 1)^2 + (2.7093\omega^3 - 2.6854\omega)^2}\right] \quad (5.6)$$

จะต้องอิเตอเรททั้งหมด 295 ครั้ง ได้ค่าต่างๆดังรูปที่ 5.4



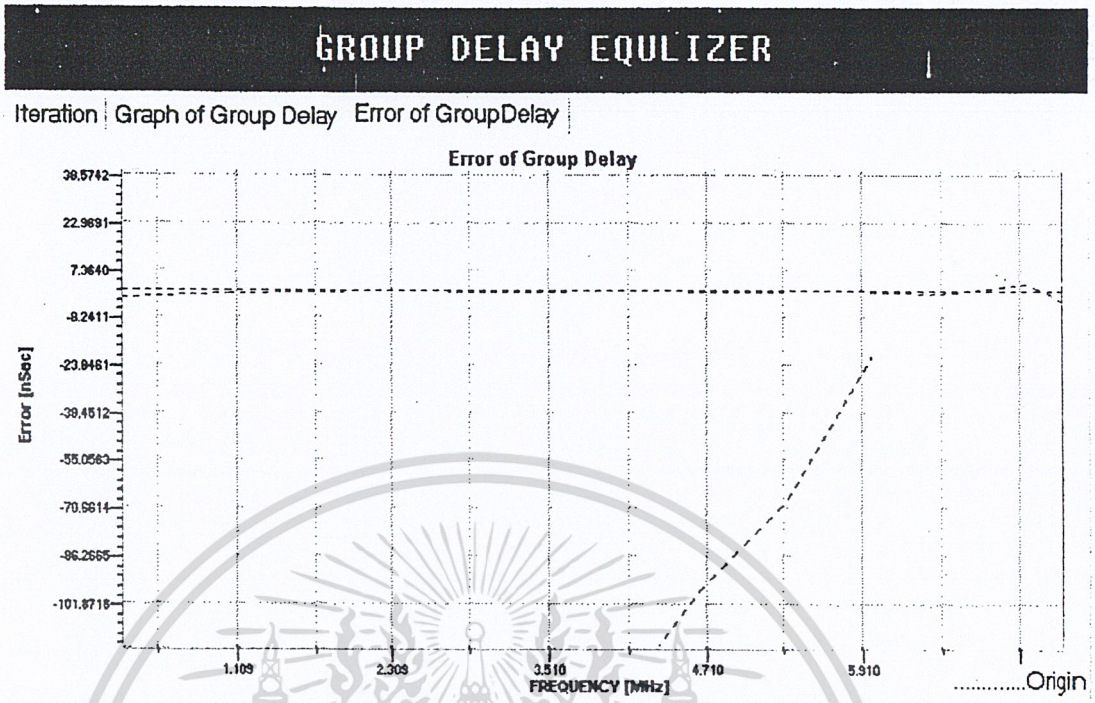
รูปที่ 5.4 แสดงหน้าต่างการอิเตอเรทของวงจร โลพาส

จะได้กราฟของกรุ๊ฟดีเลย์ ดังรูปที่ 5.5



รูปที่ 5.5 แสดงหน้าต่างกราฟของกรุ๊ฟดีเลย์ของวงจร โลพาส

จะได้ กราฟ error ของกรุ๊ฟดีเลย์ดังรูปที่ 5.6



รูปที่5.6 แสดงหน้าต่างกราฟ error ของกรุ๊ฟดีเลย์ของวงจร โลพาส

บทที่ 6
สรุปผลการทดลอง

การทดลองที่ได้ มีความละเอียดสูงมาก สามารถกำหนดผลตอบสนองของกรุ๊ฟดีเลย์ได้ตามต้องการ ขั้นแรกทำการประมาณค่าเริ่มต้นให้ตำแหน่งโพลและซีโร่ขนานไปกับแกน jw ใน s-plane แล้วนำทรานเฟอร์ฟังค์ชั่น ของวงจรที่ต้องการแก้ไขกรุ๊ฟดีเลย์มาคาสเคดกับวงจรอิควอไลเซอร์ ซึ่งก็คือการนำมาคูณกันนั่นเอง แล้วทำการหาเฟส และกรุ๊ฟดีเลย์ ของผลรวมวงจรทั้งสอง เมื่อได้สมการกรุ๊ฟดีเลย์มา จากนั้นทำการหาสมการอนุพันธ์เทียบกับ p และ q ซึ่งเป็นวิธีการของนิวตัน-ราฟสัน มาคำนวณหาค่าคลาดเคลื่อนและปรับปรุงสัมประสิทธิ์ใหม่ ค่าที่ได้จากการทดลองเป็นที่น่าพอใจ ข้อควรระวังคือสมการที่หามาต้องคำนวณให้ถูก มิฉนั้นโปรแกรมจะ error และคำนวณไม่ได้เลยต้องทำการ shut Down เครื่องใหม่ การหาสมการกรุ๊ฟดีเลย์นั้นควรแยกแฟกเตอร์ของทรานเฟอร์ฟังชั่นเพื่อไม่ให้ติดยกกำลัง จะทำให้หาสมการกรุ๊ฟดีเลย์ได้ง่ายขึ้น สมการไม่ซับซ้อนเมื่อทำการหาอนุพันธ์ในกระบวนการนิวตัน-ราฟสัน จากวิธีการดังกล่าวนี้สามารถนำค่าที่ได้ไปออกแบบทำวงจรอิควอไลเซอร์ที่สามารถกำหนดผลตอบสนองได้ตามต้องการอีกด้วย โปรแกรมที่ออกแบบมาไม่สามารถป้อนค่ากรุ๊ฟดีเลย์ได้โดยตรง ต้องเข้าไปเปลี่ยนสมการในตัวโปรแกรมเองดังที่กล่าวไว้ในการใช้งาน

เอกสารอ้างอิง

[1] Richard W.Dawiels, APPROXIMATION METHODS FOR ELECTRONICS FILTER DESIGN, McGRAW-HILL BOOK COMPANY,inc.,1974

[2] Arthur B.Williams, ELECTRONIC FILTER DESIGN HANDBOOK, McGRAW-HILL BOOK COMPANY,inc.,1981

[3] JOHNL.HILBURN and DAVIDE.JOHNSON, MANUAL OF ACTIVE FILTER DESIGN, McGRAW-HILL BOOK COMPANY, inc., 1938

[4] ไกรวุฒิ มั่นเสถียรสิน,คู่มือการเขียนโปรแกรมด้วย Delphi 4, 2528

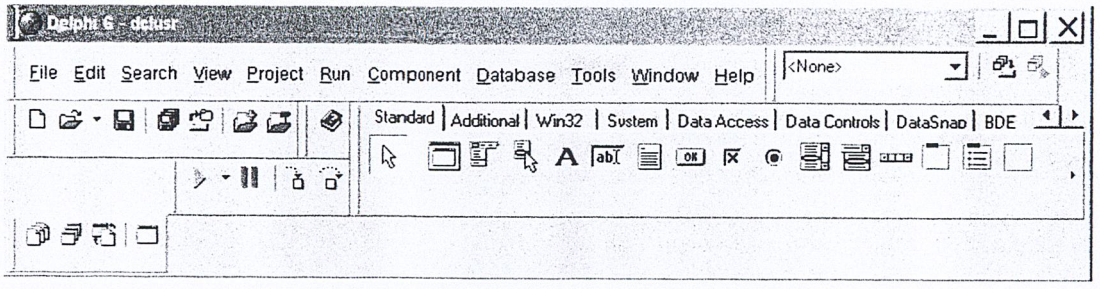
[5] ผศ.สมจิตร อาจอินทร์,หลักการเขียนโปรแกรมภาษาปาสคาล,แผนกตำรา คณะวิทยาศาสตร์ มหาวิทยาลัยขอนแก่น, 2540

[6] วรพงศ์ ตั้งศรีรัตน์,ทฤษฎีโครงข่ายไฟฟ้า,แผนกตำรา คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2544

ภาคผนวก

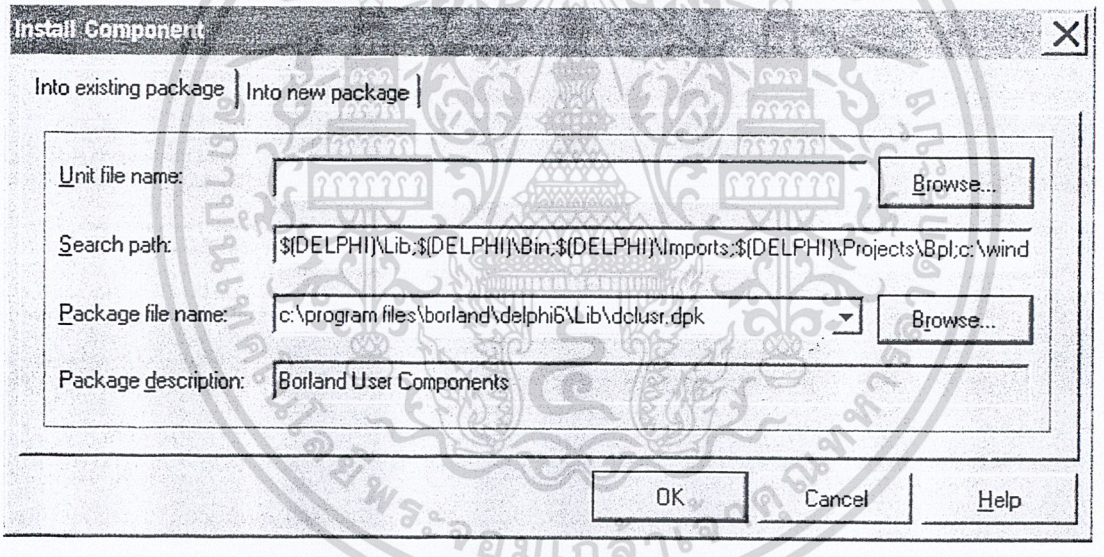
การใช้โปรแกรม

ขั้นแรกต้องทำการ ติดตั้งไฟล์ กราฟที่ใช้แสดงผลก่อน โดยเข้าไปที่ component ดังรูปที่ 1



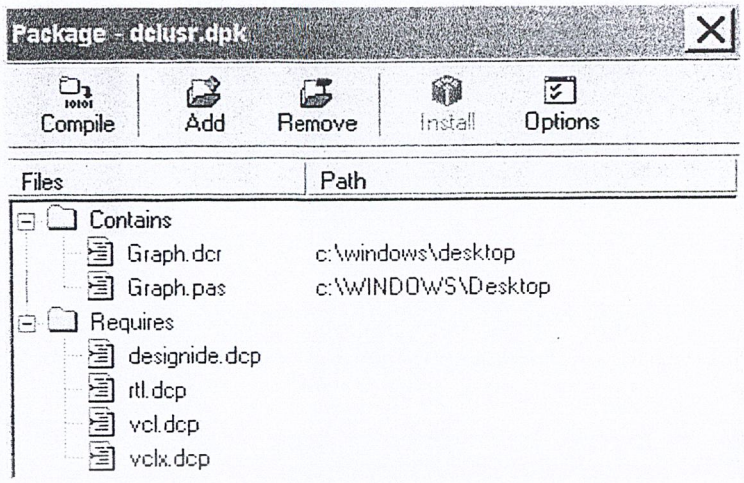
รูปที่ 1 แสดงหน้าต่างหลักของโปรแกรมเดลไฟ

โดยเลือก Install Component ดังรูปที่ 2



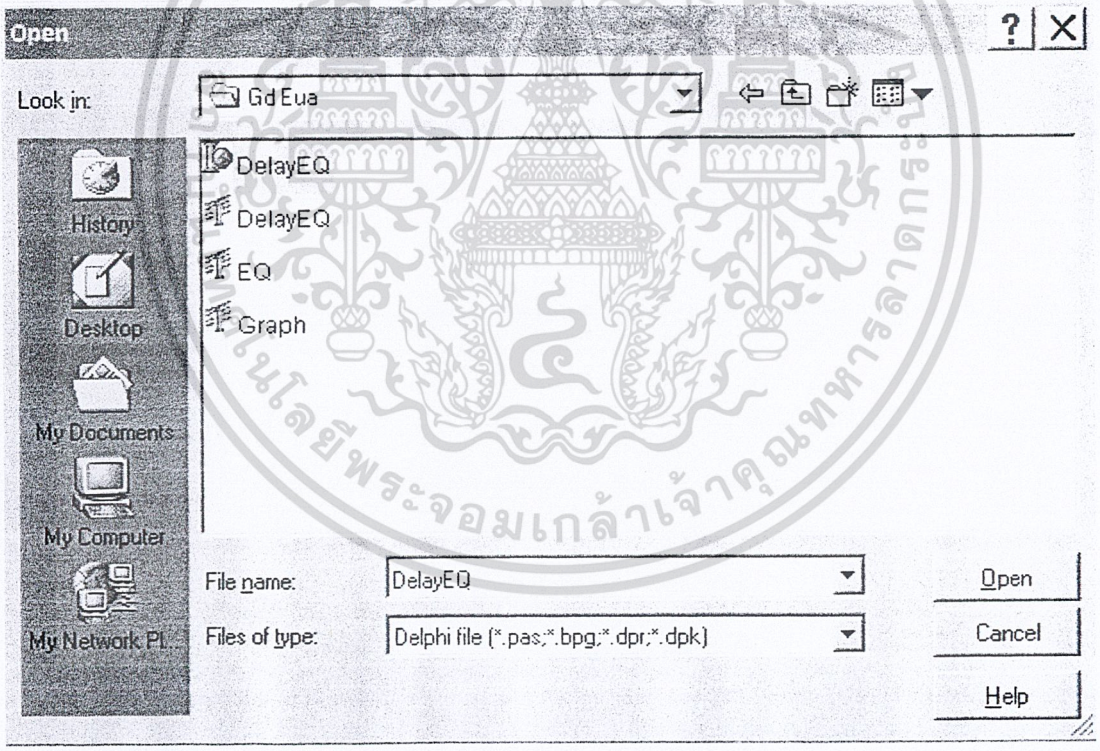
รูปที่ 2 แสดงหน้าต่าง Install Component

จากนั้น Browse หา ไฟล์ Graph.pas คลิ๊กที่ปุ่ม OK ทำการ Compile ดังรูปที่ 3 แล้วDelphi จะทำการติดตั้ง โปรแกรมสร้างกราฟ คือ กราฟของ กรุ๊ฟดีเลย์ และ กราฟ error ของกรุ๊ฟดีเลย์



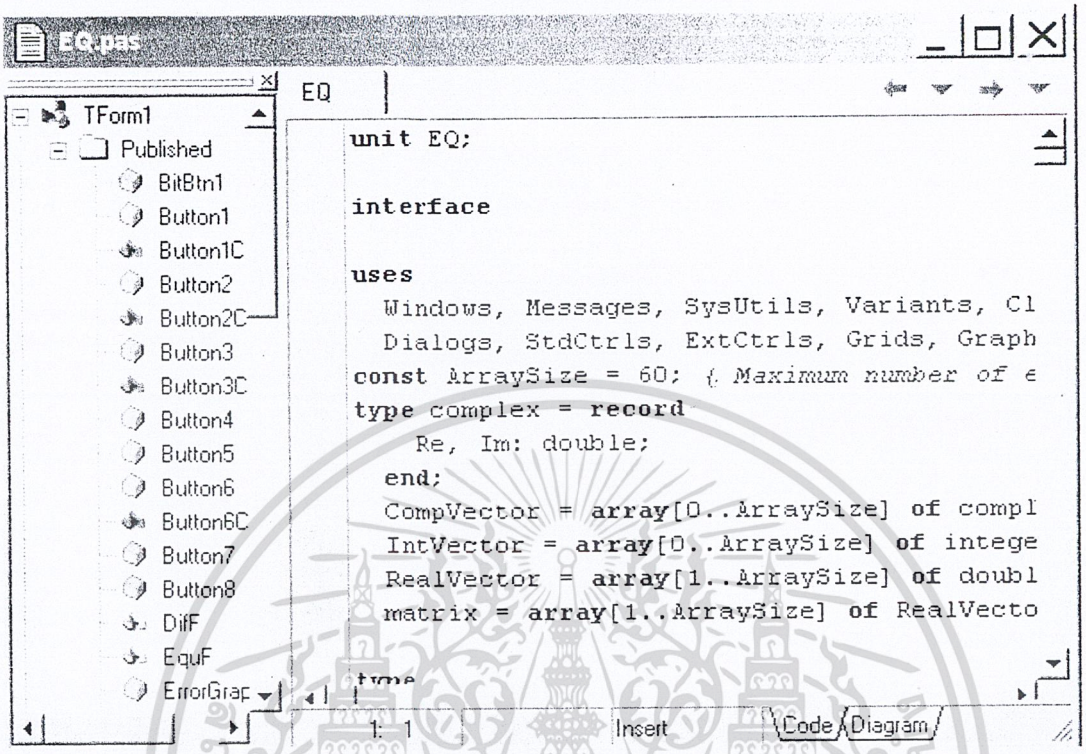
รูปที่ 3 แสดงหน้าต่างคอมไพล์ ของ Install Component

ต่อไปทำการติดตั้ง ตัวโปรแกรมที่ใช้ในการคำนวณ การอิเตอร์เรท โดยเปิดที่ File แล้ว open หา DelayEQ ดังรูปที่ 4



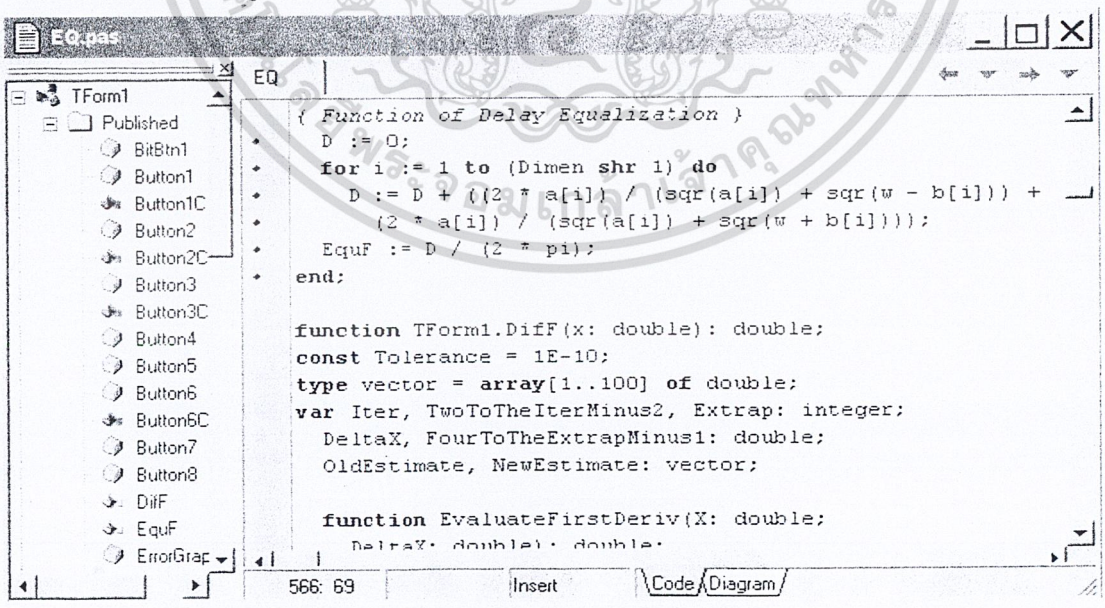
รูปที่ 4 แสดงหน้าต่างไฟล์ของตัวโปรแกรม

เมื่อติดตั้งเสร็จ Delphi จะแสดงตัวโปรแกรม ที่เขียนไว้ ดังรูปที่ 5 และแสดงหน้า ต่างอิเตอร์เรทมา ดังรูปที่ 8 ซึ่งต้องทำการปิดหน้าต่างนี้ไปก่อน เพราะยังไม่ได้ RUN โปรแกรม



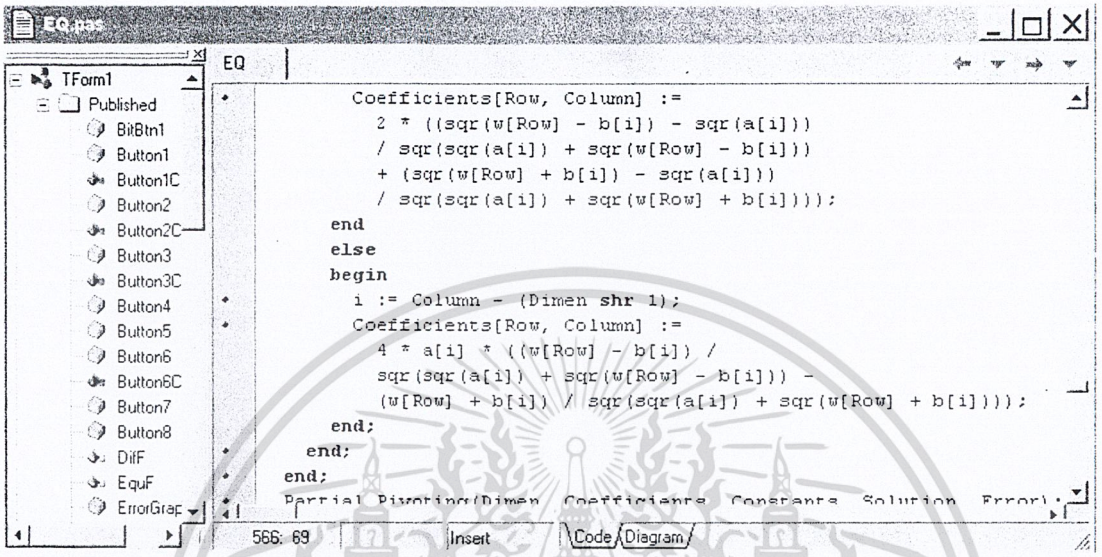
รูปที่ 5 แสดงหน้าต่างของ Code Editor

เมื่อทำการเปลี่ยนวงจรต้องเข้าไปเปลี่ยนสมการ กรุ๊ฟดีเลย์ในโปรแกรม ดังรูปที่ 6

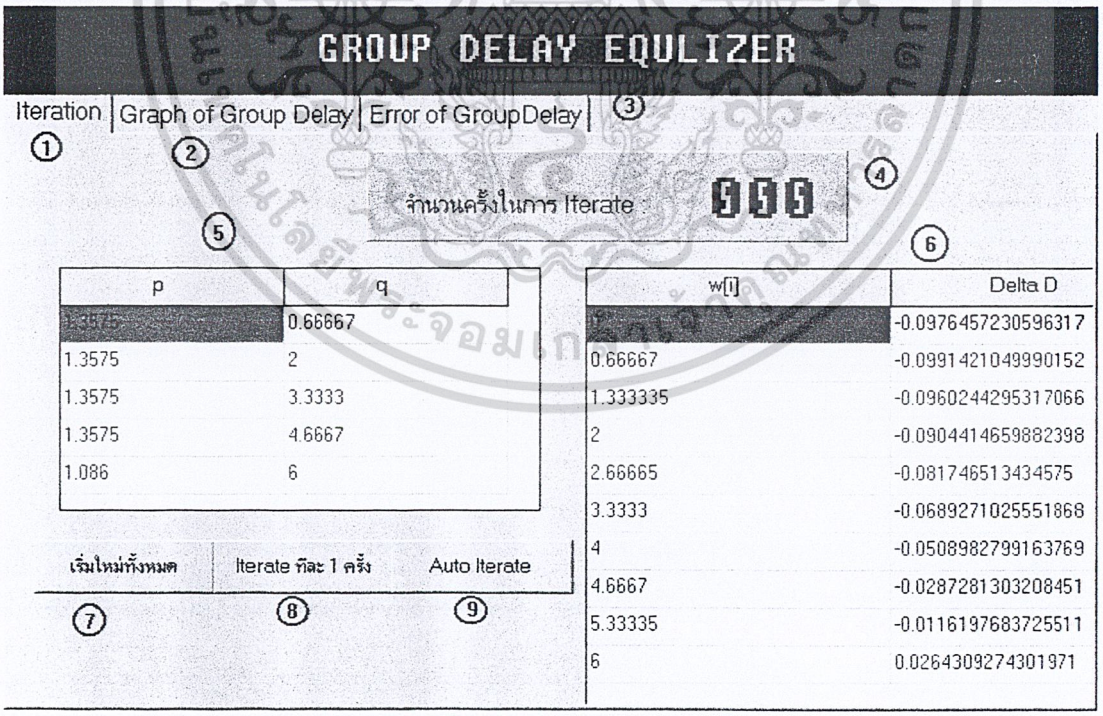


รูปที่ 6 แสดงหน้าต่าง Code Editor ที่ต้องเข้าไปเปลี่ยนสมการ

เมื่อสมการ กรุ๊ฟดีเลย์ เปลี่ยน สมการอนุพันธ์ของกรุ๊ฟดีเลย์ เทียบกับ P และ q ต้องเปลี่ยนตาม ซึ่งต้องเข้าไปเปลี่ยน โปรแกรม ดังแสดงในรูปที่ 7



รูปที่ 7 แสดงหน้าต่างโปรแกรม นิวตัน-ราฟสัน



รูปที่ 8 แสงหน้าหลักของตัวโปรแกรมที่เขียนขึ้นมา

จากรูปที่ 8 จะแสดง ปุ่มต่างๆที่ใช้ในการคำนวณ ซึ่งจะอธิบายการใช้งานดังนี้

1.ปุ่ม Iteration ปุ่มนี้จะแสดงหน้า ต่างทั้งหมด ของการอิเตอร์เรชั่น

2.ปุ่ม Graph of Group Delay ปุ่มนี้จะแสดง กราฟ ของกรุ๊ฟดีเลย์ และจำนวนครั้งในการอิเตอร์เรท

3.ปุ่ม Error of Group Delay ปุ่มนี้จะแสดงกราฟ error ของกรุ๊ฟดีเลย์ ซึ่ง กราฟที่ได้พล๊อต จากค่า Delta D เทียบ w

4.แสดงจำนวนครั้งในการอิเตอร์เรท

5.ตารางแสดงค่า p และ q

6.ตารางแสดงค่า Delta D และ w ซึ่งค่าที่ได้นี้นำไป พล๊อต กราฟ error ของกรุ๊ฟดีเลย์นั่นเอง

7.ปุ่ม เริ่มใหม่ทั้งหมด เมื่อจบการอิเตอร์เรชั่นแล้ว ถ้าอยากดูผลอีกครั้ง ปุ่มนี้จะทำการเริ่มอิเตอร์เรทใหม่

8.ปุ่ม Iterate ทีละ 1 ครั้ง หากต้องการดูผลการคำนวณค่าทีละครั้งให้กดปุ่มนี้

9.ปุ่ม Auto Iterate ปุ่มนี้จะทำการ อิเตอร์เรชั่นเองอัตโนมัต การอิเตอร์เรท จะมี 2ช่วง เมื่อจบการอิเตอร์เรชั่นครั้งแรก (*** First Iteration ***)จะมีปุ่ม “คลิกเพื่อดำเนินการต่อ” ขึ้นมาเองโดยอัตโนมัต ถ้าจะทำการ อิเตอร์เรทต่อ ให้กดปุ่มนี้ก่อน

วิธีการซูมกราฟ

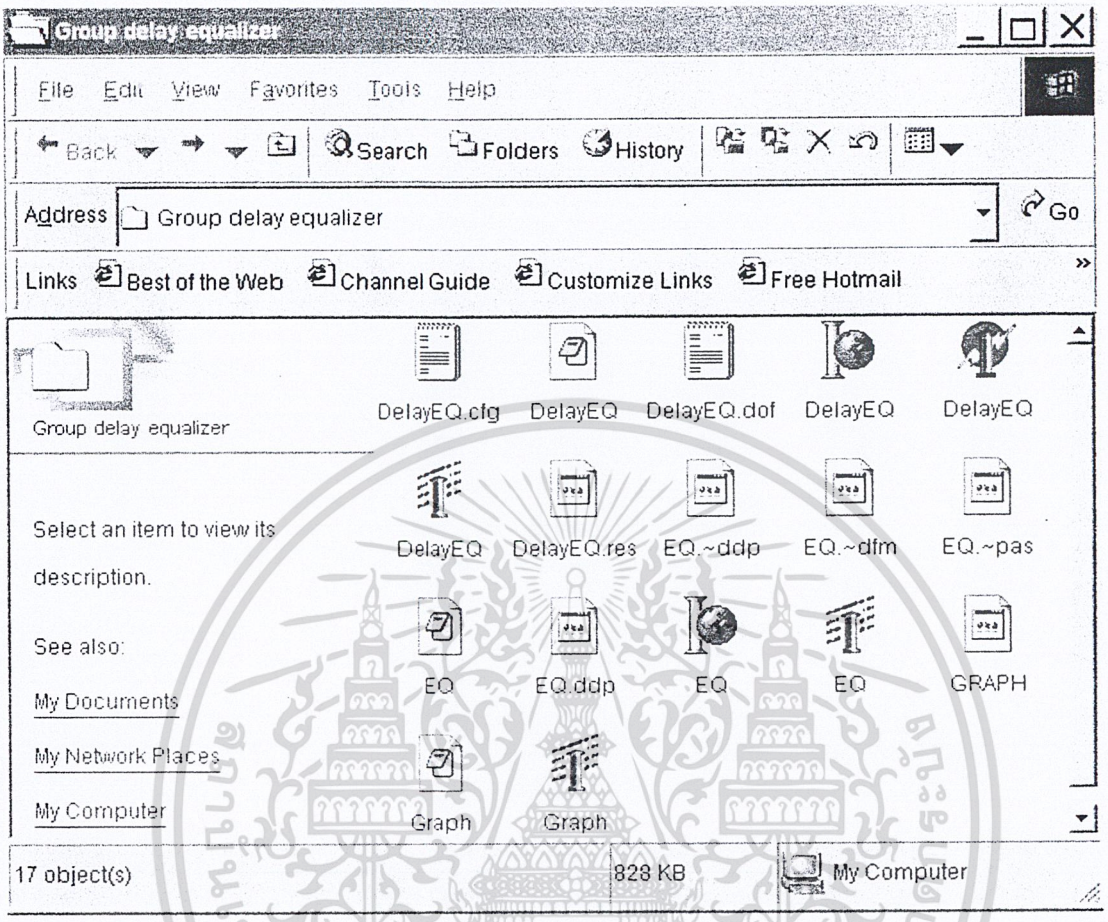
1.ซูมเข้า คลิกขวาคาไว้ แล้วเลื่อนไปทางซ้าย

2.ซูมออก คลิกขวาคาไว้ แล้วเลื่อนไปทางขวา

3.เลื่อนตาราง คลิกซ้ายคาไว้ แล้วเลื่อนไปตามต้องการ

4.กลับสู่กราฟเดิม ดับเบิลคลิก

รูปที่ 9 แสดงไฟล์ทั้งหมดที่สร้างขึ้นมา



รูปที่ 9 แสดงหน้าต่างไฟล์ทั้งหมดที่สร้างขึ้นมา

(*
Component XYGraph, Version 3.0
April 1999

U.J□ɾʁ
57078 Siegen, Germany
e-mail: ujhs@aol.com

Component XYGraph 3.0 is a versatile graph for showing 2D data:
1. + Very flexible property design.
2. + Extended control interface
   (control the graph and retrieve all curve data without writing one line
   of code by simply assigning TControlɒs to the graphɒs CONTROLS property).
3. + Powerful, flickerfree CENTERZOOM (with or without aspect ratio).
4. + Flickerfree REALPAN (single curves or graph).
5. + Editing of curve controlpoints (move, insert, delete, freeze) by mouse
   or numerical input.
6. + Free colored offset-lettering for every controlpoint.
7. + Three markstyles with scaleable size to mark important controlpoints.
8. + Relative cursor reading.
9. + Streamed writing and reading single curves or graph.
10. + DXF output of graph and curves for data exchange with CAD systems.
11. + Moveable hintpanel for additional graph-information.
12. + Example project showing some features.

Properties:
property Controls: TControls
   (all properties of type TControl can be TLabel, TStaticText, TStatusLabel
   or TPanel. They are used for data output. For example assigning a TLabel
   to the property "Mode" will display the graphɒs actual mode.)
property XOut: TControl (output) (normally the TEdit XIn)
property YOut: TControl (output) (normally the TEdit YIn)
property Mode: TControl (output)
property Curve: TControl (output)
property Item: TControl (output)
property Color: TControl (output)
property Angle: TControl (output)
  (TEdits are used for numerical inputs. (and outputs of X,Y))
property XIn: TEdit (input)
property YIn: TEdit (input)
  (TButtons are used as switches to control graph functions.)
property Clear: TButton (input)
property OpenView: TButton (input)
property OpenPan: TButton (input)
property Reset: TButton (input)
  (TRadioButtons are used to control the graphɒs operating mode.)
property ModeNone: TRadioButton (input)
property ModeMove: TRadioButton (input)
property ModeInsert: TRadioButton (input)

property ModeDelete: TRadioButton (input)
property ModeCursor: TRadioButton (input)
(TCheckBoxes are used to control graph options.)
property AspectRatio: TCheckBox (input)
property MainGrid: TCheckBox (input)
property SubGrid: TCheckBox (input)
property HintPanel: TCheckBox (input)
property ViewListBox: TCheckListBox (input)
(ViewListBox is used to control visibility of curves. Click checkmark of the curve you want to show or hide. To show/hide the ViewListBox use Button "OpenView".)
property PanListBox: TCheckListBox
(PanListBox is used to select the curve(s) you want to move in the graph. To show/hide then PanListBox use Button "OpenPan".)

property Colors: TColors
property AxisBkGnd: color of xy-axis background.
property TickColor: color of scaleticks.
property GraphBkGnd: color of graph background.
property MainGridColor: color of maingrid.
property SubGridColor: color of subgrid.

property Fonts: TFonts
property AxisScale: font of axis-scale.
property AxisTitle: font of axis title.
property GraphTitle: font of graph title.

property GraphTitle: Str32 string of graph title.

property Positions: TPositions
property XAxisLeft: left margin of xaxis.
property XAxisRight: right margin of xaxis.
property YAxisTop: top margin of yaxis.
property YAxisBottom: bottom margin of yaxis.
property TitleTop: top margin of graph title.
property TitleLeft: left margin of graph title (centered if 0).
property XAxisTitle: bottom margin of xaxis title.
property YAxisTitle: left margin of yaxis title.

property XAxis: TAxis
property Title: title of axis
property Min: min value of axis
property Max: max value of axis
property MainTicks: how many ticks with texture.
property SubTicks: how many ticks between MainTicks.
property MainTickLen: length of MainTicks.
property SubTickLen: length of SubTicks.
property Decimals: how many digits after comma.
property ShowMainGrid: flag for showing MainGrid (for color look at Colors).
property ShowSubGrid: flag for showing SubGrid (for color look at Colors).

property YAxis: TAxis read FYAxis write FYAxis;
(same as XAxis)

property MaxZoom: defines the zoomlimits (MinZoom is 1 / MaxZoom).

Control functions:
LeftButton: does REALPAN if no other function is active. If one or more curves are selected in the PanListBox, then these curves are moved instead of panning the graph. You can also set this offset by numerical input via the X/Y TEdits.
If a curve-controlpoint is active (shown by a marker), then the actual function (move, insert, delete) is performed.

RightButton: does CENTERZOOM. The point you click will be centered and then zoomed by mouse movement. If you deselect "Aspect" the X and Y axis are zoomed independent.

DoubleClick: Resets the graph (pan and zoom).

Shift: if a curve-controlpoint is marked you can freeze this point by holding down the shift-key.

How to do:
Setup the graph to your needs with the properties.
Use the

```
function MakeCurve(AName: Str32; AColor: TColor; ALineWidth: Byte;
                   APenStyle: TPenStyle; AEnabled: Boolean);
```

to create a new curve. The parameters makes the creation very flexible. Assuming AName = "test". If "test" already exists, it will be renamed to "test1" (just like Delphi does it with components).
You get a handle to the created curve. With this handle you can add new points with the

```
procedure AddPoint(AIndex: Integer; X,Y: TFloat);
```

Where AIndex is the handle to the curve. After creation of all points you can add text and/or marks to every point you want with the

```
procedure AddText(AIndex,APosition,AXOfs,AYOfs: Integer;
          const AText: Str32; AColor: TColor);
procedure AddMark(AIndex,APosition: Integer; AMarkType:
           TMarkType; AColor: TColor);
```

AIndex is the handle of the curve. APosition is the pointindex of the curve. AXOfs,AYOfs defines at witch offsets (relative to the point) where the text is displayed. Every text can be in different color. For every curve you can assign a font with the

```
  procedure SetCurveFont(AIndex: Integer; AName: TFontName;
                         ASize: Integer; AStyle: TFontStyles);
```

For every curve you can assign the size of marks with the

```
  procedure SetMarkSize(AIndex: Integer; AMarkSize: TMarkSize);
```

To add text to the HintPanel use Graph.HintPanel.Strings.Add('test');
To clear the text use Graph.HintPanel.Strings.Clear;

To use the edit functions of the graph, set Mode <> None. If moving the mouse cursor, a marker signs every controlpoint of the curve. Depending on the editmode you can move, delete or insert a point (only if the marker is visible). To freeze the actual marker, press (and hold down) the Shift-key. If EditMode is "Move", you can numerical input new point coordinates via the X,Y TEdits (if assigned).

DXF-output: you can create a DXF file with the

```
  function MakeDXF(const FileName: string; FromX1,FromY1,FromX2,FromY2,
                   ToX1,ToY1,ToX2,ToY2,TextHeight: TFloat; Decimals: Byte):
Boolean;
```

FromX1..FromY2 are the source coordinates.
ToX1..TY2 are the destination coordinates.
Decimals is the precision after comma.
All entities inside the source coordinates are transfered true to scale into the destination coordinates. Everything outside (text,scalelines etc.) are not true to scale. Does not process additional text (created with AddText) and marks (created with AddMark).

Interesting public methods are:

```
  function MakeCurve(const AName: Str32; AColor: TColor; ALineWidth: Byte;
                     APenStyle: TPenStyle; AEnabled: Boolean): Integer;
  procedure AddPoint(AIndex: Integer; X,Y: TFloat);
  procedure AddText(AIndex,APosition,AXOfs,AYOfs: Integer; const AText: Str32;
AColor: TColor);
  procedure SetCurveFont(AIndex: Integer; AName: TFontName; ASize: Integer;
AStyle: TFontStyles);
  procedure AddMark(AIndex,APosition: Integer; AMarkType: TMarkType;
AColor: TColor);
  procedure SetMarkSize(AIndex: Integer; AMarkSize: TMarkSize);
  procedure ChangePoint(AIndex,APosition: Integer; X,Y: TFloat);
  procedure DeleteCurve(AItem: Integer);
  function GetCurveHandle(AName: Str32; var H: Integer): Boolean;
  function GetCurveName(H: Integer): Str32;
```

```
  procedure SetCurveEnabled(AIndex: Integer; Value: Boolean);
  procedure GetPoint(AIndex,APosition: Integer; var X,Y: TFloat);
  procedure InsertPoint(AIndex,APosition: Integer; X,Y: TFloat);
  procedure DeletePoint(AIndex,APosition: Integer);
  procedure Reset;
  procedure ShowHintPanel(Show: Boolean);
  procedure SetXOfs(AIndex: Integer; AOfs: TFloat);
  function GetXOfs(AIndex: Integer): TFloat;
  procedure SetYOfs(AIndex: Integer; AOfs: TFloat);
  function GetYOfs(AIndex: Integer): TFloat;
  procedure CheckCurvePoints(X,Y: Integer);
  procedure ChangeCPx(Fx: TFloat);
  procedure ChangeCPy(Fy: TFloat);
  procedure ChangeCurveOfs(Ox,Oy: TFloat; Relative: Boolean);
  procedure GetCPInfo(var CPMatch: Boolean; var CPCurve,CPIndex: Integer);
  procedure SetMode(Value: TMode);
  function MakeDXF(const FileName: string; FromX1,FromY1,FromX2,FromY2,
             ToX1,ToY1,ToX2,ToY2,TextHeight: TFloat; Decimals: Byte): Boolean;
  function SaveCurveToFile(const FileName: string; Item: Integer): Boolean;
  function LoadCurveFromFile(const FileName: string): Boolean;
  function SaveGraphToFile(const FileName: string): Boolean;
  function LoadGraphFromFile(const FileName: string): Boolean;
  ---------------------------------------------------------------------

  Look at the demo project to see some other features like writing and
  reading curves or graph, or creating DXF output.

  Excuse my english - I hope you get at least the gist of it.
*)

unit
  Graph;

interface

uses
  Windows,Classes,Controls,StdCtrls,ExtCtrls,Graphics,CheckLst;
{-------------------------------------------------------------------------}

const
  MaxHintLines = 10;
{-------------------------------------------------------------------------}

type
  TFloat = Double;
  Str32 = string[32];

  TMode = (gmNone,gmMove,gmInsert,gmDelete,gmCursor);
  TGraphStyleItems = (gsMainGrid,gsSubGrid,gsHintPanel);
  TGraphStyle = set of TGraphStyleItems;
```

```
  TMarkType = (mtBox,mtCircle,mtCross);
  TMarkSize = 2..8;

  TCurveData = record //Datenstruktur f□r
SaveCurveToStream/LoadCurveFromStream
    Name: Str32;
    Enabled: Boolean;
    Color: TColor;
    LineWidth: Byte;
    PenStyle: TPenStyle;
    Points: Integer;
    Texts: Integer;
    Marks: Integer;
    XOfs: TFloat;
    YOfs: TFloat;
    FontName: Str32;
    FontSize: Integer;
    FontStyle: TFontStyles;
    MarkSize: TMarkSize;
  end;

  TGraphData = record //Datenstruktur f□r SaveGraphToFile/LoadGraphFromFile
    GraphTitle: Str32;
    Zoom: TFloat;
    MaxZoom: TFloat;
    Curves: Integer;
  end;

  PPointRec = ^TPointRec;
  TPointRec = record
          X: TFloat;
          Y: TFloat;
        end;

  PPointArray = ^TPointArray;
  TPointArray = array[0..0] of TPoint;

  PTextRec = ^TTextRec;
  TTextRec = record
    PointIndex: Integer;
    Text: Str32;
    TextColor: TColor;
    XOfs: Integer;
    YOfs: Integer;
  end;

  PMarkRec = ^TMarkRec;
  TMarkRec = record
    PointIndex: Integer;
```

```
    MarkType: TMarkType;
    MarkColor: TColor;
  end;

  TFontRec = record
    AxisScaleFontName: Str32;
    AxisScaleFontSize: Integer;
    AxisScaleFontStyle: TFontStyles;
    AxisTitleFontName: Str32;
    AxisTitleFontSize: Integer;
    AxisTitleFontStyle: TFontStyles;
    GraphTitleFontName: Str32;
    GraphTitleFontSize: Integer;
    GraphTitleFontStyle: TFontStyles;
  end;

  TDXFOut = class(TPersistent)
  private
    StringList: TStringList;
    FromXMin: TFloat;
    FromXMax: TFloat;
    FromYMin: TFloat;
    FromYMax: TFloat;
    ToXMin: TFloat;
    ToXMax: TFloat;
    ToYMin: TFloat;
    ToYMax: TFloat;
    TextHeight: TFloat;
    Decimals: Byte;
    LayerName: Str32;
  public
    constructor
  Create(AFromXMin,AFromYMin,AFromXMax,AFromYMax,AToXMin,AToYMin,
                  AToXMax,AToYMax,ATextHeight: TFloat; ADecimals: Byte);
    destructor Destroy; override;
    function FToA(F: TFloat): Str32;
    function ToX(X: TFloat): TFloat;
    function ToY(Y: TFloat): TFloat;
    procedure Header;
    procedure Trailer;
    procedure SetLayer(const Name: Str32);
    procedure Line(X1,Y1,Z1,X2,Y2,Z2: TFloat);
    procedure Point(X,Y,Z: TFloat);
    procedure StartPolyLine(Closed: Boolean);
    procedure Vertex(X,Y,Z: TFloat);
    procedure EndPolyLine;
    procedure DText(X,Y,Z,Height,Angle: TFloat; const Txt: Str32);
    procedure Layer;
    procedure StartPoint(X,Y,Z: TFloat);
    procedure EndPoint(X,Y,Z: TFloat);
```

```
  procedure AddText(const Txt: Str32);
end;

TXYGraph = class;

TControls = class(TPersistent)
private
  Graph: TXYGraph;
  FXOut: TControl;
  FYOut: TControl;
  FMode: TControl;
  FCurve: TControl;
  FItem: TControl;
  FColor: TControl;
  FAngle: TControl;
  FXIn: TEdit;
  FYIn: TEdit;
  FClear: TButton;
  FOpenView: TButton;
  FOpenPan: TButton;
  FReset: TButton;
  FModeNone: TRadioButton;
  FModeMove: TRadioButton;
  FModeInsert: TRadioButton;
  FModeDelete: TRadioButton;
  FModeCursor: TRadioButton;
  FAspectRatio: TCheckBox;
  FMainGrid: TCheckBox;
  FSubGrid: TCheckBox;
  FHintPanel: TCheckBox;
  FViewListBox: TCheckListBox;
  FPanListBox: TCheckListBox;
protected
  procedure SetControl(Index: Integer; Value: TControl);
  procedure SetEdit(Index: Integer; Value: TEdit);
  procedure SetButton(Index: Integer; Value: TButton);
  procedure SetRadioButton(Index: Integer; Value: TRadioButton);
  procedure SetCheckBox(Index: Integer; Value: TCheckBox);
  procedure SetListBox(Index: Integer; Value: TCheckListBox);
public
  constructor Create(AGraph: TXYGraph);
published
  property XOut: TControl index 0 read FXOut write SetControl;
  property YOut: TControl index 1 read FYOut write SetControl;
  property Mode: TControl index 2 read FMode write SetControl;
  property Curve: TControl index 3 read FCurve write SetControl;
  property Item: TControl index 4 read FItem write SetControl;
  property Color: TControl index 5 read FColor write SetControl;
  property Angle: TControl index 6 read FAngle write SetControl;
```

```
    property XIn: TEdit index 0 read FXIn write SetEdit;
    property YIn: TEdit index 1 read FYIn write SetEdit;

    property Clear: TButton index 0 read FClear write SetButton;
    property OpenView: TButton index 1 read FOpenView write SetButton;
    property OpenPan: TButton index 2 read FOpenPan write SetButton;
    property Reset: TButton index 3 read FReset write SetButton;

    property ModeNone: TRadioButton index 0 read FModeNone write
SetRadioButton;
    property ModeMove: TRadioButton index 1 read FModeMove write
SetRadioButton;
    property ModeInsert: TRadioButton index 2 read FModeInsert write
SetRadioButton;
    property ModeDelete: TRadioButton index 3 read FModeDelete write
SetRadioButton;
    property ModeCursor: TRadioButton index 4 read FModeCursor write
SetRadioButton;

    property AspectRatio: TCheckBox index 0 read FAspectRatio write SetCheckBox;
    property MainGrid: TCheckBox index 1 read FMainGrid write SetCheckBox;
    property SubGrid: TCheckBox index 2 read FSubGrid write SetCheckBox;
    property HintPanel: TCheckBox index 3 read FHintPanel write SetCheckBox;

    property ViewListBox: TCheckListBox index 0 read FViewListBox write
SetListBox;
    property PanListBox: TCheckListBox index 1 read FPanListBox write SetListBox;
  end;

  THintPanel = class(TCustomPanel)
  private
    FStrings: TStringList;
    Graph: TXYGraph;
    Moving: Boolean;
    Start: Boolean;
    MouseX: Integer;
    MouseY: Integer;
  protected
    procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X,Y: Integer);
override;
    procedure MouseMove(Shift: TShiftState; X,Y: Integer); override;
    procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X,Y: Integer);
override;
    procedure Loaded; override;
    procedure NewBounds;
    procedure DoStringsChange(Sender: TObject);
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure Paint; override;
```

```
  property Strings: TStringList read FStrings write FStrings;
end;

TPositions = class(TPersistent)
private
  FXAxisLeft: Integer;
  FXAxisRight: Integer;
  FYAxisTop: Integer;
  FYAxisBottom: Integer;
  FTitleTop: Integer;
  FTitleLeft: Integer;
  FXAxisTitle: Integer;
  FYAxisTitle: Integer;
  FOnChange: TNotifyEvent;
protected
  procedure SetInteger(Index,Value: Integer);
public
  constructor Create;
  property OnChange: TNotifyEvent read FOnChange write FOnChange default nil;
published
  property XAxisLeft: Integer index 0 read FXAxisLeft write SetInteger;
  property XAxisRight: Integer index 1 read FXAxisRight write SetInteger;
  property YAxisTop: Integer index 2 read FYAxisTop write SetInteger;
  property YAxisBottom: Integer index 3 read FYAxisBottom write SetInteger;
  property TitleTop: Integer index 4 read FTitleTop write SetInteger;
  property TitleLeft: Integer index 5 read FTitleLeft write SetInteger;
  property XAxisTitle: Integer index 6 read FXAxisTitle write SetInteger;
  property YAxisTitle: Integer index 7 read FYAxisTitle write SetInteger;
end;

TFonts = class(TPersistent)
private
  FAxisScale: TFont;
  FAxisTitle: TFont;
  FGraphTitle: TFont;
  FOnChange: TNotifyEvent;
protected
  procedure SetFont(Index: Integer; Value: TFont);
public
  constructor Create;
  destructor Destroy; override;
  property OnChange: TNotifyEvent read FOnChange write FOnChange;
published
  property AxisScale: TFont index 0 read FAxisScale write SetFont;
  property AxisTitle: TFont index 1 read FAxisTitle write SetFont;
  property GraphTitle: TFont index 2 read FGraphTitle write SetFont;
end;

TColors = class(TPersistent)
private
```

```
  FAxisBkGnd: TColor;
  FTickColor: TColor;
  FGraphBkGnd: TColor;
  FMainGridColor: TColor;
  FSubGridColor: TColor;
  FOnChange: TNotifyEvent;
protected
  procedure SetColor(Index: Integer; Value: TColor);
public
  constructor Create;
  property OnChange: TNotifyEvent read FOnChange write FOnChange;
published
  property AxisBkGnd: TColor index 0 read FAxisBkGnd write SetColor;
  property TickColor: TColor index 1 read FTickColor write SetColor;
  property GraphBkGnd: TColor index 2 read FGraphBkGnd write SetColor;
  property MainGridColor: TColor index 3 read FMainGridColor write SetColor;
  property SubGridColor: TColor index 4 read FSubGridColor write SetColor;
end;

TAxis = class(TPersistent)
private
  FTitle: Str32;                    {Beschriftung Achsentitel}
  FLength: Integer;                 {Achsenl□nge}
  FMin: TFloat;                     {Anfangswert auf Achse}
  FMax: TFloat;                     {Endwert auf Achse}
  FMinSave: TFloat;                 {Anfangswert auf Achse}
  FMaxSave: TFloat;                 {Endwert auf Achse}
  FZoom: TFloat;
  FMainTicks: Byte;                 {Anzahl Hauptteilungen}
  FSubTicks: Byte;                  {Anzahl Subteilungen}
  FMainTickLen: Byte;               {L□nge der Hauptskalenstriche}
  FSubTickLen: Byte;                {L□nge der Subskalenstriche}
  FDecimals: Byte;      {Anzahl Nachkommastellen f□r Beschriftung auf Achse}
  FShowMainGrid: Boolean;
  FShowSubGrid: Boolean;
  FFactor: TFloat;                  {Wertfaktor f□r 1 Pixel auf Achse}
  FValuePerMainTick: TFloat;
  FValuePerPixel: TFloat;
  FTicks: Integer;
  FPixelsPerSubTick: TFloat;
  FPan: Integer;
  FPanSubTicks: Integer;
  FOnChange: TNotifyEvent;
protected
  procedure SetTitle(const Value: Str32);
  procedure SetLength(Value: Integer);
  procedure SetFloat(Index: Integer; Value: TFloat);
  procedure SetMax(Value: TFloat);
  procedure SetByte(Index: Integer; Value: Byte);
```

```
  procedure SetBoolean(Index: Integer; Value: Boolean);
public
  constructor Create;
  procedure CalcAxis;
  function Value(APosition: Integer): TFloat;
  function Pixel(APosition: TFloat): Integer;
  procedure SetMinMax(AMin,AMax: TFloat);
  procedure SetZoom(Value: TFloat);
  procedure SetCenter(C: TFloat);
  procedure SetLeftBottom(L: TFloat);
  procedure SetRightTop(R: TFloat);
  function GetCenter: TFloat;
  property Length: Integer read FLength write SetLength default 200;
  property ValuePerMainTick: TFloat read FValuePerMainTick;
  property ValuePerPixel: TFloat read FValuePerPixel;
  property PixelsPerSubTick: TFloat read FPixelsPerSubTick;
  property Pan: Integer read FPan write FPan;
  property PanSubTicks: Integer read FPanSubTicks write FPanSubTicks;
  property Ticks: Integer read FTicks;
  property Zoom: TFloat read FZoom write FZoom;
  property OnChange: TNotifyEvent read FOnChange write FOnChange;
published
  property Title: Str32 read FTitle write SetTitle;
  property Min: TFloat index 0 read FMin write SetFloat;
  property Max: TFloat index 1 read FMax write SetFloat;
  property MainTicks: Byte index 0 read FMainTicks write SetByte;
  property SubTicks: Byte index 1 read FSubTicks write SetByte;
  property MainTickLen: Byte index 2 read FMainTickLen write SetByte;
  property SubTickLen: Byte index 3 read FSubTickLen write SetByte;
  property Decimals: Byte index 4 read FDecimals write SetByte;
  property ShowMainGrid: Boolean index 0 read FShowMainGrid write SetBoolean;
  property ShowSubGrid: Boolean index 1 read FShowSubGrid write SetBoolean;
end;

TCurve = class(TPersistent)
private
  FPoints: TList;
  FTexts: TList;
  FMarks: TList;
  FFont: TFont;
  FName: Str32;
  FEnabled: Boolean;
  FColor: TColor;
  FLineWidth: Byte;
  FPenStyle: TPenStyle;
  FXOfs: TFloat;
  FYOfs: TFloat;
  FMarkSize: TMarkSize;
  PPoint: PPointRec;
  PText: PTextRec;
```

```
    PMark: PMarkRec;
  public
    constructor Create;
    destructor Destroy; override;
    procedure AddPoint(Ax,Ay: TFloat);
    procedure AddText(APointIndex,AXOfs,AYOfs: Integer; const AText: Str32;
AColor: TColor);
    procedure AddMark(APointIndex: Integer; AMarkType: TMarkType; AColor:
TColor);
    procedure GetPoint(AIndex: Integer; var Ax,Ay: TFloat);
    procedure ChangePoint(AIndex: Integer; Ax,Ay: TFloat);
    procedure InsertPoint(AIndex: Integer; Ax,Ay: TFloat);
    procedure DeletePoint(AIndex: Integer);
  public
    property Name: Str32 read FName write FName;
    property Enabled: Boolean read FEnabled write FEnabled;
    property Color: TColor read FColor write FColor;
    property LineWidth: Byte read FLineWidth write FLineWidth;
    property PenStyle: TPenStyle read FPenStyle write FPenStyle;
    property XOfs: TFloat read FXOfs write FXOfs;
    property YOfs: TFloat read FYOfs write FYOfs;
    property MarkSize: TMarkSize read FMarkSize write FMarkSize;
  end;

  TXYGraph = class(TCustomPanel)
    procedure SetBounds(ALeft,ATop,AWidth,AHeight: Integer); override;
  private
    FXAxis: TAxis;
    FYAxis: TAxis;
    FColors: TColors;
    FPositions: TPositions;
    FFonts: TFonts;
    FCurve: TCurve;
    FCurveList: TList;
    FHintPanel: THintPanel;
    FControls: TControls;
    FMode: TMode;
    FGraphTitle: Str32;
    FZoom: TFloat;
    FMaxZoom: TFloat;

    DrawBmp: TBitMap;
    DXFOut: TDXFOut;
    MouseX: Integer;
    MouseY: Integer;
    CPBmp: TBitMap;
    CPRect: TRect;
    CPMatch: Boolean;              {Flag f□r Kontrollpunkterkennung}
    CPCurve: Integer;              {Index f□r Kurve des Kontrollpunktes}
    LastCPCurve: Integer;          {Index f□r Kurve des Kontrollpunktes}
```

```
  CPIndex: Integer;              {Index f□r Kontrollpunkt in der Kurve}
  LastCPIndex: Integer;          {Index f□r Kontrollpunkt in der Kurve}
  CPx: TFloat;                   {X-Wert des Kontrollpunktes}
  CPy: TFloat;                   {Y-Wert des Kontrollpunktes}

  IsLoaded: Boolean;                          {Flag f□r Loaded}
  BoundsChanged: Boolean;
  ZoomSave: TFloat;
  Freeze: Boolean;
  ZoomAspectRatio: Boolean;
  PanCurves: Boolean;
  HClip: HRgn;
 protected
  procedure Loaded; override;
  procedure DrawXAxis;
  procedure DrawYAxis;
  procedure OnChangePaint(Sender: TObject);
  procedure DoButtonClick(Sender: TObject);
  procedure DoRadioButtonClick(Sender: TObject);
  procedure DoCheckBoxClick(Sender: TObject);
  procedure DoListBoxClickCheck(Sender: TObject);
  procedure DoXEditExit(Sender: TObject);
  procedure DoYEditExit(Sender: TObject);

  procedure DoPan(Dx,Dy: Integer);
  procedure DoZoom(Dx,Dy: Integer);
  procedure DoMouse(X,Y: Integer);
  procedure SetMeasureCursor(X,Y: Integer);
  procedure DoMeasureCursor(X,Y: Integer);
  procedure DoMove(Dx,Dy: Integer);
  procedure DoCheckCP(X,Y: Integer);

  procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X,Y: Integer);
override;
  procedure MouseMove(Shift: TShiftState; X,Y: Integer); override;
  procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X,Y: Integer);
override;
  procedure DblClicked(Sender: TObject);
  procedure SetGraphTitle(const Value: Str32);
  procedure SetEditEnable(Value: Boolean);
  procedure OutMode(const Mode: Str32);
  procedure OutCurve(const Curve: Str32);
  procedure OutItem(Item: Integer);
  procedure OutColor(Color: TColor);
  procedure OutAngle(A: TFloat);
  procedure OutXY(Fx,Fy: TFloat);
  procedure ClearMarkBox;
  procedure DrawMarkBox;
  procedure DrawMark(ACanvas: TCanvas; MarkType: TMarkType;
            MarkColor: TColor; MarkSize: TMarkSize; X,Y: Integer);
```

```
 public
   constructor Create(AOwner: TComponent); override;
   destructor Destroy; override;
   procedure Paint; override;
   procedure Notification(Component: TComponent; Operation: TOperation);
override;
   function MakeCurve(const AName: Str32; AColor: TColor; ALineWidth: Byte;
                APenStyle: TPenStyle; AEnabled: Boolean): Integer;
   procedure AddPoint(AIndex: Integer; X,Y: TFloat);
   procedure AddText(AIndex,APosition,AXOfs,AYOfs: Integer; const AText: Str32;
AColor: TColor);
   procedure SetCurveFont(AIndex: Integer; AName: TFontName; ASize: Integer;
AStyle: TFontStyles);
   procedure AddMark(AIndex,APosition: Integer; AMarkType: TMarkType;
AColor: TColor);
   procedure SetMarkSize(AIndex: Integer; AMarkSize: TMarkSize);
   procedure ChangePoint(AIndex,APosition: Integer; X,Y: TFloat);
   procedure DeleteCurve(AItem: Integer);
   function GetCurveHandle(AName: Str32; var H: Integer): Boolean;
   function GetCurveName(H: Integer): Str32;
   procedure SetCurveEnabled(AIndex: Integer; Value: Boolean);
   procedure GetPoint(AIndex,APosition: Integer; var X,Y: TFloat);
   procedure InsertPoint(AIndex,APosition: Integer; X,Y: TFloat);
   procedure DeletePoint(AIndex,APosition: Integer);
   procedure Reset;
   procedure ShowHintPanel(Show: Boolean);
   procedure SetXOfs(AIndex: Integer; AOfs: TFloat);
   function GetXOfs(AIndex: Integer): TFloat;
   procedure SetYOfs(AIndex: Integer; AOfs: TFloat);
   function GetYOfs(AIndex: Integer): TFloat;
   procedure CheckCurvePoints(X,Y: Integer);
   procedure ChangeCPx(Fx: TFloat);   {X-Wert Kontrollpunkt ฦnderung von auЫen}
   procedure ChangeCPy(Fy: TFloat);   {Y-Wert Kontrollpunkt ฦnderung von auЫen}
   procedure ChangeCurveOfs(Ox,Oy: TFloat; Relative: Boolean);
   procedure GetCPInfo(var CPMatch: Boolean; var CPCurve,CPIndex: Integer);
   procedure SetZoom(Value: TFloat);
   procedure SetMode(Value: TMode);
   function GetMaxPoints: Integer;
   function XAxisPixel(Value: TFloat): Integer;
   function YAxisPixel(Value: TFloat): Integer;

   function MakeDXF(const FileName: string; FromX1,FromY1,FromX2,FromY2,
            ToX1,ToY1,ToX2,ToY2,TextHeight: TFloat; Decimals: Byte): Boolean;
   procedure DXFAxis;
   procedure DXFCurves;

   function SaveCurveToStream(FileStream: TFileStream; Item: Integer): Boolean;
   function LoadCurveFromStream(FileStream: TFileStream): Boolean;
   function SaveCurveToFile(const FileName: string; Item: Integer): Boolean;
```

```
    function LoadCurveFromFile(const FileName: string): Boolean;
    function SaveGraphToFile(const FileName: string): Boolean;
    function LoadGraphFromFile(const FileName: string): Boolean;

    property HintPanel: THintPanel read FHintPanel write FHintPanel;
    property Mode: TMode read FMode write SetMode;
    property Zoom: TFloat read FZoom write SetZoom;
  published
    property Align;
    property Anchors;
    property Hint;
    property ShowHint;
    property OnClick;
    property OnKeyPress;
    property OnKeyDown;
    property OnMouseMove;
    property OnMouseDown;
    property OnMouseUp;

    property Colors: TColors read FColors write FColors;
    property Fonts: TFonts read FFonts write FFonts;
    property GraphTitle: Str32 read FGraphTitle write SetGraphTitle;
    property Positions: TPositions read FPositions write FPositions;
    property XAxis: TAxis read FXAxis write FXAxis;
    property YAxis: TAxis read FYAxis write FYAxis;
    property Controls: TControls read FControls write FControls;
    property MaxZoom: TFloat read FMaxZoom write FMaxZoom;
  end;
{------------------------------------------------------------------------------}

procedure Register;
function AtoF(S: Str32; var F: TFloat): Boolean;
function InRange(Test,Min,Max: Integer): Boolean;
function Angle(X1,Y1,X2,Y2: TFloat): TFloat;      {0§-360§ gegen Uhrzeigersinn}
procedure TextOutRotate(ACanvas: TCanvas; X,Y: Integer; Ang: Word; S: Str32);

implementation
{------------------------------------------------------------------------------}

uses
  SysUtils,Forms,Dialogs,ComCtrls;
{------------------------------------------------------------------------------}

procedure Register;
begin
  RegisterComponents('Udo',[TXYGraph]);
end;
{------------------------------------------------------------------------------}

function AtoF(S: Str32; var F: TFloat): Boolean;
```

```
var
  Code: Integer;
begin
  Code:=Pos(',',S);
  if Code > 0 then S[Code]:='.';
  Val(S,F,Code);
  Result:=Code = 0;
end;
{-------------------------------------------------------------------------}

function InRange(Test,Min,Max: Integer): Boolean;
begin
  Result:=(Test >= Min) and (Test <= Max);
end;
{-------------------------------------------------------------------------}

function Angle(X1,Y1,X2,Y2: TFloat): TFloat;          {Steigung = 0..90ß}
var                                                   {Gefälle = 0..-90ß}
  Dx: TFloat;
  Dy: TFloat;
begin
  Result:=0;
  Dx:=X2 - X1;
  Dy:=Y2 - Y1;
  if Dx <> 0 then Result:=ArcTan(Dy / Dx) / Pi * 180;
  if Dx < 0 then Result:=-Result;
end;
{-------------------------------------------------------------------------}

procedure TextOutRotate(ACanvas: TCanvas; X,Y: Integer; Ang: Word; S: Str32);
var
  LogRec: TLogFont;
  OldFontHandle: HFont;
  NewFontHandle: HFont;
begin
  GetObject(ACanvas.Font.Handle,SizeOf(LogRec),@LogRec);
  LogRec.lfEscapement:=Ang;
  NewFontHandle:=CreateFontIndirect(LogRec);
  OldFontHandle:=SelectObject(ACanvas.Handle,NewFontHandle);
  ACanvas.TextOut(X,Y,S);
  NewFontHandle:=SelectObject(ACanvas.Handle,OldFontHandle);
  DeleteObject(NewFontHandle);
end;
{-------------------------------------------------------------------------}

constructor TCurve.Create;
begin
  inherited Create;
  FPoints:=TList.Create;
```

```
  FTexts:=TList.Create;
  FMarks:=TList.Create;
  FFont:=TFont.Create;
  FFont.Name:='small font';
  FFont.Size:=7;
  FFont.Style:=[];
  FMarkSize:=4;
  FEnabled:=True;
  FColor:=clWhite;
  FLineWidth:=1;
  FPenStyle:=psSolid;
  FXOfs:=0.0;
  FYOfs:=0.0;
end;
{--------------------------------------------------------------------------}

destructor TCurve.Destroy;
var
  I: Integer;
begin
  for I:=0 to Pred(FPoints.Count) do FreeMem(FPoints.Items[I],SizeOf(TPointRec));
  FPoints.Free;
  for I:=0 to Pred(FTexts.Count) do FreeMem(FTexts.Items[I],SizeOf(TTextRec));
  FTexts.Free;
  for I:=0 to Pred(FMarks.Count) do FreeMem(FMarks.Items[I],SizeOf(TMarkRec));
  FMarks.Free;
  FFont.Free;
  inherited Destroy;
end;
{--------------------------------------------------------------------------}

procedure TCurve.AddPoint(Ax,Ay: TFloat);
begin
  GetMem(PPoint,SizeOf(TPointRec));
  PPoint^.X:=Ax;
  PPoint^.Y:=Ay;
  FPoints.Add(PPoint);
end;
{--------------------------------------------------------------------------}

procedure TCurve.AddText(APointIndex,AXOfs,AYOfs: Integer; const AText: Str32;
AColor: TColor);
begin
  GetMem(PText,SizeOf(TTextRec));
  PText^.PointIndex:=APointIndex;
  PText^.XOfs:=AXOfs;
  PText^.YOfs:=AYOfs;
  PText^.Text:=AText;
  PText^.TextColor:=AColor;
  FTexts.Add(PText);
```

```
end;
{-------------------------------------------------------------------------------}

procedure TCurve.AddMark(APointIndex: Integer; AMarkType: TMarkType;
AColor: TColor);
begin
  GetMem(PMark,SizeOf(TMarkRec));
  PMark^.PointIndex:=APointIndex;
  PMark^.MarkType:=AMarkType;
  PMark^.MarkColor:=AColor;
  FMarks.Add(PMark);
end;
{-------------------------------------------------------------------------------}

procedure TCurve.GetPoint(AIndex: Integer; var Ax,Ay: TFloat);
begin
  if InRange(AIndex,0,Pred(FPoints.Count)) then
  begin
    PPoint:=FPoints.Items[AIndex];
    Ax:=PPoint^.X + FXOfs;
    Ay:=PPoint^.Y + FYOfs;
  end;
end;
{-------------------------------------------------------------------------------}

procedure TCurve.ChangePoint(AIndex: Integer; Ax,Ay: TFloat);
begin
  if InRange(AIndex,0,Pred(FPoints.Count)) then
  begin
    PPoint:=FPoints.Items[AIndex];
    PPoint^.X:=Ax - FXOfs;
    PPoint^.Y:=Ay - FYOfs;
  end;
end;
{-------------------------------------------------------------------------------}

procedure TCurve.InsertPoint(AIndex: Integer; Ax,Ay: TFloat);
begin
  if AIndex > -1 then
  begin
    GetMem(PPoint,SizeOf(TPointRec));
    PPoint^.X:=Ax;
    PPoint^.Y:=Ay;
    FPoints.Insert(AIndex,PPoint);
  end;
end;
{-------------------------------------------------------------------------------}

procedure TCurve.DeletePoint(AIndex: Integer);
begin
```

```
 if InRange(AIndex,0,Pred(FPoints.Count)) then
 begin
   FreeMem(FPoints.Items[AIndex],SizeOf(TPointRec));
   FPoints.Delete(AIndex);
 end;
end;
{--------------------------------------------------------------------------}

constructor TPositions.Create;
begin
 inherited Create;
 FXAxisLeft:=60;
 FXAxisRight:=15;
 FYAxisTop:=30;
 FYAxisBottom:=50;
 FTitleTop:=5;
 FTitleLeft:=0;
 FXAxisTitle:=20;
 FYAxisTitle:=5;
 FOnChange:=nil;
end;
{--------------------------------------------------------------------------}

procedure TPositions.SetInteger(Index,Value: Integer);
begin
  case Index of
    0 : FXAxisLeft:=Value;
    1 : FXAxisRight:=Value;
    2 : FYAxisTop:=Value;
    3 : FYAxisBottom:=Value;
    4 : FTitleTop:=Value;
    5 : FTitleLeft:=Value;
    6 : FXAxisTitle:=Value;
    7 : FYAxisTitle:=Value;
  end;
  if Assigned(FOnChange) then FOnChange(Self);
end;
{--------------------------------------------------------------------------}

constructor TFonts.Create;
begin
  inherited Create;
  FAxisScale:=TFont.Create;
  FAxisScale.Name:='small fonts';
  FAxisScale.Size:=7;
  FAxisScale.Color:=clNavy;

  FAxisTitle:=TFont.Create;
  FAxisTitle.Name:='arial';
  FAxisTitle.Size:=8;
```

```
 FAxisTitle.Style:=[fsBold];
 FAxisTitle.Color:=clMaroon;

 FGraphTitle:=TFont.Create;
 FGraphTitle.Name:='arial';
 FGraphTitle.Size:=10;
 FGraphTitle.Style:=[fsBold];
 FGraphTitle.Color:=clMaroon;
 FOnChange:=nil;
end;
{--------------------------------------------------------------------------}

destructor TFonts.Destroy;
begin
 FAxisScale.Free;
 FAxisTitle.Free;
 FGraphTitle.Free;
 inherited Destroy;
end;
{--------------------------------------------------------------------------}

procedure TFonts.SetFont(Index: Integer; Value: TFont);
begin
 case Index of
  0 : FAxisScale.Assign(Value);
  1 : FAxisTitle.Assign(Value);
  2 : FGraphTitle.Assign(Value);
 end;
 if Assigned(FOnChange) then FOnChange(Self);
end;
{--------------------------------------------------------------------------}

constructor TColors.Create;
begin
 inherited Create;
 FAxisBkGnd:=clSilver;
 FTickColor:=clBlack;
 FGraphBkGnd:=clBlack;
 FMainGridColor:=clGray;
 FSubGridColor:=clGray;
 FOnChange:=nil;
end;
{--------------------------------------------------------------------------}

procedure TColors.SetColor(Index: Integer; Value: TColor);
begin
 case Index of
  0: FAxisBkGnd:=Value;
  1: FTickColor:=Value;
  2: FGraphBkGnd:=Value;
```

```
    3: FMainGridColor:=Value;
    4: FSubGridColor:=Value;
   end;
   if Assigned(FOnChange) then FOnChange(Self);
end;
{-------------------------------------------------------------------------}

constructor TAxis.Create;
begin
  inherited Create;
  FTitle:='Axis-Title';
  FLength:=200;
  FMin:=0;
  FMax:=10;
  FMinSave:=FMin;
  FMaxSave:=FMax;
  FZoom:=1.0;
  FMainTicks:=5;
  FSubTicks:=5;
  FMainTickLen:=10;
  FSubTickLen:=5;
  FDecimals:=2;
  FPan:=0;
  FPanSubTicks:=0;
  FShowMainGrid:=True;
  FShowSubGrid:=False;
  FOnChange:=nil;
  CalcAxis;
end;
{-------------------------------------------------------------------------}

procedure TAxis.CalcAxis;
begin
  FValuePerMainTick:=(FMax - FMin) / FMainTicks;
  FFactor:=FLength / (FMax - FMin);
  FTicks:=FMainTicks * FSubTicks;
  FPixelsPerSubTick:=FLength / FTicks;
  FValuePerPixel:=FValuePerMainTick / (FSubTicks * FPixelsPerSubTick);
end;
{-------------------------------------------------------------------------}

function TAxis.Value(APosition: Integer): TFloat;
begin
  Result:=FMin + (FValuePerPixel * (APosition - FPan));
end;
{-------------------------------------------------------------------------}

function TAxis.Pixel(APosition: TFloat): Integer;
begin
  Result:=FPan + Round((APosition - FMin) * FFactor);
```

```
end;
{-------------------------------------------------------------------------------}

procedure TAxis.SetZoom(Value: TFloat);
var
  Zoom,Dif: TFloat;
begin
  Dif:=FMax - FMin;
  Zoom:=Dif / (Value * FZoom);
  FMin:=FMin - Dif + Zoom;
  FMax:=FMax + Dif - Zoom;
  CalcAxis;
end;
{-------------------------------------------------------------------------------}

procedure TAxis.SetCenter(C: TFloat);
var
  Dif: TFloat;
begin
  Dif:=(FMax - FMin) / 2;
  FMin:=C - Dif;
  FMax:=C + Dif;
  CalcAxis;
end;
{-------------------------------------------------------------------------------}

function TAxis.GetCenter: TFloat;
begin
  Result:=FMin + ((FMax - FMin) / 2);
end;
{-------------------------------------------------------------------------------}

procedure TAxis.SetLeftBottom(L: TFloat);
var
  Dif: TFloat;
begin
  Dif:=FMax - FMin;
  FMin:=L;
  FMax:=FMin + Dif;
  CalcAxis;
end;
{-------------------------------------------------------------------------------}

procedure TAxis.SetRightTop(R: TFloat);
var
  Dif: TFloat;
begin
  Dif:=FMax - FMin;
  FMax:=R;
  FMin:=FMax - Dif;
```

```
 CalcAxis;
end;
{-------------------------------------------------------------------------}

procedure TAxis.SetMinMax(AMin,AMax: TFloat);
begin
 if (AMin < FMax) and (AMax > FMin) then
 begin
  FMin:=AMin;
  FMax:=AMax;
 end;
end;
{-------------------------------------------------------------------------}

procedure TAxis.SetTitle(const Value: Str32);
begin
 if FTitle <> Value then
 begin
  FTitle:=Value;
  if Assigned(FOnChange) then FOnChange(Self);
 end;
end;
{-------------------------------------------------------------------------}

procedure TAxis.SetLength(Value: Integer);
begin
 if FLength <> Value then
 begin
  FLength:=Value;
  CalcAxis;
  if Assigned(FOnChange) then FOnChange(Self);
 end;
end;
{-------------------------------------------------------------------------}

procedure TAxis.SetFloat(Index: Integer; Value: TFloat);
begin
  case Index of
    0: if (Value <> FMin) and (Value < FMax) then
       begin
         FMin:=Value;
         FMinSave:=FMin;
         CalcAxis;
         if Assigned(FOnChange) then FOnChange(Self);
       end;
    1: if (Value <> FMax) and (Value > FMin) then
       begin
         FMax:=Value;
         FMaxSave:=Value;
         CalcAxis;
```

```
      if Assigned(FOnChange) then FOnChange(Self);
    end;
  end;
end;
{-------------------------------------------------------------------------}

procedure TAxis.SetMax(Value: TFloat);
begin
end;
{-------------------------------------------------------------------------}

procedure TAxis.SetByte(Index: Integer; Value: Byte);
begin
  case Index of
    0 : if Value > 0 then FMainTicks:=Value;
    1 : FSubTicks:=Value;
    2 : FMainTickLen:=Value;
    3 : FSubTickLen:=Value;
    4 : if Value < 5 then FDecimals:=Value;
  end;
  CalcAxis;
  if Assigned(FOnChange) then FOnChange(Self);
end;
{-------------------------------------------------------------------------}

procedure TAxis.SetBoolean(Index: Integer; Value: Boolean);
begin
  case Index of
    0 : FShowMainGrid:=Value;
    1 : FShowSubGrid:=Value;
  end;
  if Assigned(FOnChange) then FOnChange(Self);
end;
{-------------------------------------------------------------------------}

constructor TXYGraph.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);

  IsLoaded:=False;
  BoundsChanged:=False;
  ZoomAspectRatio:=True;
  PanCurves:=False;
  CPMatch:=False;
  FZoom:=1.0;
  MaxZoom:=5.0;
  SetBounds(Left,Top,400,300);

  FXAxis:=TAxis.Create;
  FYAxis:=TAxis.Create;
```

```
  FColors:=TColors.Create;
  FFonts:=TFonts.Create;
  FPositions:=TPositions.Create;
  FCurveList:=TList.Create;
  DrawBmp:=TBitMap.Create;

  FControls:=TControls.Create(Self);

  FHintPanel:=THintPanel.Create(Self);
  FHintPanel.Parent:=Self;
  FHintPanel.Visible:=True;

  FXAxis.OnChange:=OnChangePaint;
  FYAxis.OnChange:=OnChangePaint;
  FFonts.OnChange:=OnChangePaint;
  FColors.OnChange:=OnChangePaint;
  FPositions.OnChange:=OnChangePaint;

  OnDblClick:=DblClicked;

  FGraphTitle:='Graph-Title';
  XAxis.Title:='X-Axis-Title';
  YAxis.Title:='Y-Axis-Title';
end;
{------------------------------------------------------------------------}

destructor TXYGraph.Destroy;
var
  I: Integer;
begin
  FXAxis.Free;
  FYAxis.Free;
  FPositions.Free;
  FColors.Free;
  FFonts.Free;
  for I:=0 to Pred(FCurveList.Count) do
  begin
    FCurve:=FCurveList.Items[I];
    FCurve.Free;
  end;
  FCurveList.Free;
  DrawBmp.Free;
  FHintPanel.Free;
  FControls.Free;

  inherited Destroy;
end;
{------------------------------------------------------------------------}

procedure TXYGraph.Loaded;
```

```
begin
  inherited Loaded;
  XAxis.FMinSave:=XAxis.FMin;
  XAxis.FMaxSave:=XAxis.FMax;
  YAxis.FMinSave:=YAxis.FMin;
  YAxis.FMaxSave:=YAxis.FMax;

  XAxis.CalcAxis;
  YAxis.CalcAxis;

  ZoomSave:=Zoom;

  SetMode(gmNone);
  IsLoaded:=True;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.SetBounds(ALeft,ATop,AWidth,AHeight: Integer);
begin
  inherited SetBounds(ALeft,ATop,AWidth,AHeight);
  BoundsChanged:=True;
end;
{--------------------------------------------------------------------------}

function TXYGraph.XAxisPixel(Value: TFloat): Integer;
begin
  Result:=Positions.XAxisLeft + XAxis.Pixel(Value);
end;
{--------------------------------------------------------------------------}

function TXYGraph.YAxisPixel(Value: TFloat): Integer;
begin
  Result:=Height - Positions.YAxisBottom - YAxis.Pixel(Value);
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.Paint;
var
  R: TRect;
  H,I,J: Integer;
  X,Y: TFloat;
  Size: Word;
  PA: PPointArray;
  PText: PTextRec;
  PMark: PMarkRec;
begin
  if BoundsChanged then
  begin
    XAxis.Length:=Width - Positions.XAxisLeft - Positions.XAxisRight;
    YAxis.Length:=Height - Positions.YAxisTop - Positions.YAxisBottom;
```

```
  BoundsChanged:=False;
end;

if CPMatch or (FMode = gmCursor) then ClearMarkBox;

DrawBmp.Width:=Width;
DrawBmp.Height:=Height;
DrawBmp.Canvas.Pen.Width:=1;

DrawBmp.Canvas.Brush.Color:=FColors.FGraphBkGnd;
DrawBmp.Canvas.FillRect(Rect(FPositions.XAxisLeft,FPositions.YAxisTop,
          Width - Positions.XAxisRight,Height - Positions.YAxisBottom));
DrawBmp.Canvas.Brush.Color:=FColors.AxisBkGnd;
DrawBmp.Canvas.FillRect(Rect(0,0,Positions.XAxisLeft,Height));

DrawBmp.Canvas.FillRect(Rect(Positions.XAxisLeft,0,Width,Positions.YAxisTop));
DrawBmp.Canvas.FillRect(Rect(0,Height - Positions.YAxisBottom,Width,Height));
DrawBmp.Canvas.FillRect(Rect(Width -
Positions.XAxisRight,Positions.YAxisTop,Width,Height -
FPositions.YAxisBottom));

DrawBmp.Canvas.Brush.Color:=clGray;
R:=Rect(0,0,Width,Height);
DrawBmp.Canvas.FrameRect(R);
InflateRect(R,-1,-1);
DrawBmp.Canvas.Brush.Color:=clWhite;
DrawBmp.Canvas.FrameRect(R);
DrawBmp.Canvas.Brush.Style:=bsClear;

if Length(FGraphTitle) > 0 then
begin
  DrawBmp.Canvas.Font:=FFonts.FGraphTitle;
  if Positions.TitleLeft = 0 then
    DrawBmp.Canvas.TextOut(Width div 2 -
DrawBmp.Canvas.TextWidth(FGraphTitle) div 2,
          Positions.TitleTop,FGraphTitle)
  else
DrawBmp.Canvas.TextOut(Positions.TitleLeft,Positions.TitleTop,FGraphTitle);
end;

DrawXAxis;
DrawYAxis;

HClip:=CreateRectRgn(Positions.XAxisLeft,Positions.YAxisTop,
          Width - Positions.XAxisRight + 1,Height - Positions.YAxisBottom +
1);
SelectClipRgn(DrawBmp.Canvas.Handle,HClip);
Size:=GetMaxPoints * SizeOf(TPointArray);
GetMem(PA,Size);
```

```
for H:=0 to Pred(FCurveList.Count) do
begin
  FCurve:=FCurveList.Items[H];
  if FCurve.Enabled and (FCurve.FPoints.Count > 0) then
  begin
    DrawBmp.Canvas.Pen.Color:=FCurve.Color;
    DrawBmp.Canvas.Pen.Style:=FCurve.PenStyle;
    DrawBmp.Canvas.Pen.Width:=FCurve.LineWidth;
    J:=Pred(FCurve.FPoints.Count);
    for I:=0 to J do
    begin
      FCurve.GetPoint(I,X,Y);
      PA^[I].x:=XAxisPixel(X);
      PA^[I].y:=YAxisPixel(Y);
    end;
    DrawBmp.Canvas.PolyLine(Slice(PA^,Succ(J)));
    for I:=0 to Pred(FCurve.FTexts.Count) do
    begin
      PText:=FCurve.FTexts.Items[I];
      DrawBmp.Canvas.Font:=FCurve.FFont;
      DrawBmp.Canvas.Font.Color:=PText^.TextColor;
      DrawBmp.Canvas.Brush.Style:=bsClear;
      FCurve.GetPoint(PText^.PointIndex,X,Y);
      DrawBmp.Canvas.TextOut(XAxisPixel(X) + PText^.XOfs,
                  YAxisPixel(Y) + PText^.YOfs,PText^.Text);
    end;
    for I:=0 to Pred(FCurve.FMarks.Count) do
    begin
      PMark:=FCurve.FMarks.Items[I];
      FCurve.GetPoint(PMark^.PointIndex,X,Y);
      DrawMark(DrawBmp.Canvas,PMark^.MarkType,PMark^.MarkColor,
          FCurve.FMarkSize,XAxisPixel(X),YAxisPixel(Y));
    end;
  end;
end;
FreeMem(PA,Size);
DeleteObject(HClip);

DrawBmp.Canvas.Pen.Style:=psSolid;
R:=ClientRect;
Canvas.CopyRect(R,DrawBmp.Canvas,R);

if CPMatch or (FMode = gmCursor) then DrawMarkBox;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.OnChangePaint(Sender: TObject);
begin
  if Sender = FPositions then
  begin
```

```
  XAxis.Length:=Width - FPositions.XAxisLeft - FPositions.XAxisRight;
  YAxis.Length:=Height - FPositions.YAxisTop - FPositions.YAxisBottom;
 end;
 Application.ProcessMessages;
 if IsLoaded then Paint;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DrawXAxis;
var
 I,X,Y: Integer;
 Pos: Integer;
 VPos: Integer;
 S: string[12];
begin
 DrawBmp.Canvas.Font:=FFonts.AxisScale;
 DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
 Y:=0;
 VPos:=Height - FPositions.YAxisBottom;
 FXAxis.PanSubTicks:=Round(FXAxis.Pan / FXAxis.PixelsPerSubTick);
 if FXAxis.Pan > 0 then
 begin
  if FXAxis.PanSubTicks >= FXAxis.SubTicks then
  begin
   FXAxis.Pan:=FXAxis.Pan - Round(FXAxis.PanSubTicks *
FXAxis.PixelsPerSubTick);
   FXAxis.PanSubTicks:=0;
   FXAxis.SetMinMax(FXAxis.Min - FXAxis.ValuePerMainTick,
                    FXAxis.Max - FXAxis.ValuePerMainTick);
  end;
  for X:=-FXAxis.PanSubTicks to FXAxis.Ticks - FXAxis.PanSubTicks do
  begin
   Pos:=FPositions.XAxisLeft + Round(FXAxis.FPixelsPerSubTick * X) +
FXAxis.Pan;
   if (Pos >= FPositions.XAxisLeft) and (Pos <= Width - FPositions.XAxisRight)
then
    begin
     DrawBmp.Canvas.MoveTo(Pos,VPos);
     if X mod FXAxis.SubTicks = 0 then
     begin
      DrawBmp.Canvas.LineTo(Pos,VPos + FXAxis.MainTickLen);
      S:=FloatToStrF(FXAxis.Min + (Y *
FXAxis.ValuePerMainTick),ffFixed,7,FXAxis.Decimals);
      for I:=1 to Length(S) do
      if S[I] = ' ' then Delete(S,I,1);
      I:=DrawBmp.Canvas.TextWidth(S);
      DrawBmp.Canvas.TextOut(Pos - I div 2,VPos + FXAxis.MainTickLen,S);
      if FXAxis.ShowMainGrid then
      begin
       DrawBmp.Canvas.Pen.Color:=FColors.MainGridColor;
```

```
        DrawBmp.Canvas.MoveTo(Pos,FPositions.YAxisTop);
        DrawBmp.Canvas.LineTo(Pos,Height - FPositions.YAxisBottom);
        DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
       end;
      end
      else
      begin
       DrawBmp.Canvas.LineTo(Pos,VPos + FXAxis.SubTickLen);
       if FXAxis.ShowSubGrid then
       begin
        DrawBmp.Canvas.Pen.Color:=FColors.SubGridColor;
        DrawBmp.Canvas.MoveTo(Pos,FPositions.YAxisTop);
        DrawBmp.Canvas.LineTo(Pos,Height - FPositions.YAxisBottom);
        DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
       end;
      end;
     end;
     if X mod FXAxis.SubTicks = 0 then Inc(Y);
    end;
   end
   else
   begin
    if FXAxis.PanSubTicks <= -FXAxis.SubTicks then
    begin
     FXAxis.Pan:=FXAxis.Pan - Round(FXAxis.PanSubTicks *
FXAxis.PixelsPerSubTick);
     FXAxis.PanSubTicks:=0;
     FXAxis.SetMinMax(FXAxis.Min + FXAxis.ValuePerMainTick,
                      FXAxis.Max + FXAxis.ValuePerMainTick);
    end;
    for X:=FXAxis.Ticks - FXAxis.PanSubTicks downto 0 - FXAxis.PanSubTicks do
    begin
     Pos:=FPositions.XAxisLeft + Round(FXAxis.FPixelsPerSubTick * X) +
FXAxis.Pan;
     if (Pos >= FPositions.XAxisLeft) and (Pos <= Width - FPositions.XAxisRight)
then
     begin
      DrawBmp.Canvas.MoveTo(Pos,VPos);
      if X mod FXAxis.SubTicks = 0 then
      begin
       DrawBmp.Canvas.LineTo(Pos,VPos + FXAxis.MainTickLen);
       S:=FloatToStrF(FXAxis.Max - (Y *
FXAxis.ValuePerMainTick),ffFixed,7,FXAxis.Decimals);
       for I:=1 to Length(S) do if S[I] = ' ' then Delete(S,I,1);
       I:=DrawBmp.Canvas.TextWidth(S);
       DrawBmp.Canvas.TextOut(Pos - I div 2,VPos + FXAxis.MainTickLen,S);
       if FXAxis.ShowMainGrid then
       begin
        DrawBmp.Canvas.Pen.Color:=FColors.MainGridColor;
        DrawBmp.Canvas.MoveTo(Pos,FPositions.YAxisTop);
```

```
          DrawBmp.Canvas.LineTo(Pos,Height - FPositions.YAxisBottom);
          DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
        end;
      end
      else
      begin
        DrawBmp.Canvas.LineTo(Pos,VPos + FXAxis.SubTickLen);
        if FXAxis.ShowSubGrid then
        begin
          DrawBmp.Canvas.Pen.Color:=FColors.SubGridColor;
          DrawBmp.Canvas.MoveTo(Pos,FPositions.YAxisTop);
          DrawBmp.Canvas.LineTo(Pos,Height - FPositions.YAxisBottom);
          DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
        end;
      end;
    end;
    if X mod FXAxis.SubTicks = 0 then Inc(Y);
  end;
 end;
 DrawBmp.Canvas.Pen.Color:=FColors.GraphBkGnd;
 DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,VPos);
 DrawBmp.Canvas.LineTo(Width - FPositions.XAxisRight,VPos);
 DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,FPositions.YAxisTop);
 DrawBmp.Canvas.LineTo(Width - FPositions.XAxisRight,FPositions.YAxisTop);

 I:=DrawBmp.Canvas.TextWidth(XAxis.Title);
 X:=FPositions.XAxisLeft + XAxis.Length div 2 - I div 2;
 DrawBmp.Canvas.Font:=FFonts.AxisTitle;
 DrawBmp.Canvas.TextOut(X,Height - FPositions.FXAxisTitle,XAxis.Title);
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DrawYAxis;
var
 I,H,X,Y: Integer;
 Pos: Integer;
 S: string[12];
begin
 DrawBmp.Canvas.Font:=FFonts.AxisScale;
 DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
 H:=DrawBmp.Canvas.TextHeight('0');
 Y:=0;
 FYAxis.PanSubTicks:=Round(FYAxis.Pan / FYAxis.PixelsPerSubTick);
 if FYAxis.Pan > 0 then
 begin
   if FYAxis.PanSubTicks >= FYAxis.SubTicks then
   begin
     FYAxis.Pan:=FYAxis.Pan - Round(FYAxis.PanSubTicks *
FYAxis.PixelsPerSubTick);
     FYAxis.PanSubTicks:=0;
```

```
    FYAxis.SetMinMax(FYAxis.Min - FYAxis.ValuePerMainTick,
                     FYAxis.Max - FYAxis.ValuePerMainTick);
   end;
   for X:=-FYAxis.PanSubTicks to FYAxis.Ticks - FYAxis.PanSubTicks do
   begin
    Pos:=Height - FPositions.YAxisBottom - Round(FYAxis.FPixelsPerSubTick * X)
- FYAxis.Pan;
    if (Pos >= FPositions.YAxisTop) and (Pos <= Height - FPositions.YAxisBottom)
then
    begin
     DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,Pos);
     if X mod FYAxis.SubTicks = 0 then
     begin
      DrawBmp.Canvas.LineTo(FPositions.XAxisLeft - FYAxis.MainTickLen,Pos);
      S:=FloatToStrF(FYAxis.Min + (Y *
FYAxis.ValuePerMainTick),ffFixed,7,FYAxis.Decimals);
      for I:=1 to Length(S) do
      if S[I] = ' ' then Delete(S,I,1);
      I:=DrawBmp.Canvas.TextWidth(S);
      DrawBmp.Canvas.TextOut(FPositions.XAxisLeft - FYAxis.MainTickLen -
I,Pos - H div 2,S);
      if FYAxis.ShowMainGrid and (Pos <> FPositions.YAxisTop)
                  and (Pos <> Height - FPositions.YAxisBottom) then
      begin
       DrawBmp.Canvas.Pen.Color:=FColors.MainGridColor;
       DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,Pos);
       DrawBmp.Canvas.LineTo(Width - FPositions.XAxisRight,Pos);
       DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
      end;
     end
     else
     begin
      DrawBmp.Canvas.LineTo(FPositions.XAxisLeft - FYAxis.SubTickLen,Pos);
      if FYAxis.ShowSubGrid then
      begin
       DrawBmp.Canvas.Pen.Color:=FColors.SubGridColor;
       DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,Pos);
       DrawBmp.Canvas.LineTo(Width - FPositions.XAxisRight,Pos);
       DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
      end;
     end;
    end;
    if X mod FYAxis.SubTicks = 0 then Inc(Y);
   end;
  end
  else
  begin
   if FYAxis.PanSubTicks <= -FYAxis.SubTicks then
   begin
```

```
    FYAxis.Pan:=FYAxis.Pan - Round(FYAxis.PanSubTicks *
FYAxis.PixelsPerSubTick);
    FYAxis.PanSubTicks:=0;
    FYAxis.SetMinMax(FYAxis.Min + FYAxis.ValuePerMainTick,
                FYAxis.Max + FYAxis.ValuePerMainTick);
   end;
   for X:=FYAxis.Ticks - FYAxis.PanSubTicks downto 0 - FYAxis.PanSubTicks do
   begin
    Pos:=Height - FPositions.YAxisBottom - Round(FYAxis.FPixelsPerSubTick * X)
- FYAxis.Pan;
    if (Pos >= FPositions.YAxisTop) and (Pos <= Height - FPositions.YAxisBottom)
then
     begin
      DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,Pos);
      if X mod FYAxis.SubTicks = 0 then
      begin
       DrawBmp.Canvas.LineTo(FPositions.XAxisLeft - FYAxis.MainTickLen,Pos);
       S:=FloatToStrF(FYAxis.Max - (Y *
FYAxis.ValuePerMainTick),ffFixed,7,FYAxis.Decimals);
       for I:=1 to Length(S) do
       if S[I] = ' ' then Delete(S,I,1);
       I:=DrawBmp.Canvas.TextWidth(S);
       DrawBmp.Canvas.TextOut(FPositions.XAxisLeft - FYAxis.MainTickLen -
I,Pos - H div 2,S);
       if FYAxis.ShowMainGrid and (Pos <> FPositions.YAxisTop)
                  and (Pos <> Height - FPositions.YAxisBottom) then
       begin
        DrawBmp.Canvas.Pen.Color:=FColors.MainGridColor;
        DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,Pos);
        DrawBmp.Canvas.LineTo(Width - FPositions.XAxisRight,Pos);
        DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
       end;
      end
      else
      begin
       DrawBmp.Canvas.LineTo(FPositions.XAxisLeft - FYAxis.SubTickLen,Pos);
       if FYAxis.ShowSubGrid then
       begin
        DrawBmp.Canvas.Pen.Color:=FColors.SubGridColor;
        DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,Pos);
        DrawBmp.Canvas.LineTo(Width - FPositions.XAxisRight,Pos);
        DrawBmp.Canvas.Pen.Color:=FColors.FTickColor;
       end;
      end;
     end;
     if X mod FYAxis.SubTicks = 0 then Inc(Y);
    end;
   end;
   DrawBmp.Canvas.Pen.Color:=FColors.GraphBkGnd;
```

```
  DrawBmp.Canvas.MoveTo(FPositions.XAxisLeft,Height -
FPositions.YAxisBottom);
  DrawBmp.Canvas.LineTo(FPositions.XAxisLeft,FPositions.YAxisTop - 1);
  DrawBmp.Canvas.MoveTo(Width - FPositions.XAxisRight,Height -
FPositions.YAxisBottom);
  DrawBmp.Canvas.LineTo(Width - FPositions.XAxisRight,FPositions.YAxisTop -
1);

  I:=DrawBmp.Canvas.TextWidth(YAxis.Title);
  Y:=Height - FPositions.YAxisBottom - YAxis.Length div 2 + I div 2;
  DrawBmp.Canvas.Font:=FFonts.AxisTitle;
  TextOutRotate(DrawBmp.Canvas,FPositions.FYAxisTitle,Y,900,YAxis.Title);
end;
{-------------------------------------------------------------------------------}

function TXYGraph.GetMaxPoints: Integer;
var
  I,Max: Integer;
begin
  Max:=0;
  for I:=0 to Pred(FCurveList.Count) do
  begin
    FCurve:=FCurveList.Items[I];
    if FCurve.FPoints.Count > Max then Max:=FCurve.FPoints.Count;
  end;
  Result:=Max;
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.SetEditEnable(Value: Boolean);
begin
  if Assigned(FControls.XOut) then FControls.XOut.Enabled:=Value;
  if Assigned(FControls.YOut) then FControls.YOut.Enabled:=Value;
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.OutMode(const Mode: Str32);
begin
  if Assigned(FControls.FMode) then
  begin
    if (FControls.FMode is TPanel) then TPanel(FControls.FMode).Caption:=Mode;
    if (TObject(FControls.FMode) is TStatusPanel) then
TStatusPanel(FControls.FMode).Text:=Mode;
    if (FControls.FMode is TLabel) then TLabel(FControls.FMode).Caption:=Mode;
    if (FControls.FMode is TStaticText) then
TStaticText(FControls.FMode).Caption:=Mode;
  end;
end;
{-------------------------------------------------------------------------------}
```

```
procedure TXYGraph.OutCurve(const Curve: Str32);
begin
  if Assigned(FControls.FCurve) then
  begin
    if (FControls.FCurve is TPanel) then TPanel(FControls.FCurve).Caption:=Curve;
    if (TObject(FControls.FCurve) is TStatusPanel) then
TStatusPanel(FControls.FCurve).Text:=Curve;
    if (FControls.FCurve is TLabel) then TLabel(FControls.FCurve).Caption:=Curve;
    if (FControls.FCurve is TStaticText) then
TStaticText(FControls.FCurve).Caption:=Curve;
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.OutItem(Item: Integer);
var
  Text: Str32;
begin
  if Item > -1 then Text:=IntToStr(Item) else Text:='';
  if Assigned(FControls.FItem) then
  begin
    if (FControls.FItem is TPanel) then TPanel(FControls.FItem).Caption:=Text;
    if (TObject(FControls.FItem) is TStatusPanel) then
TStatusPanel(FControls.FItem).Text:=Text;
    if (FControls.FItem is TLabel) then TLabel(FControls.FItem).Caption:=Text;
    if (FControls.FItem is TStaticText) then
TStaticText(FControls.FItem).Caption:=Text;
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.OutColor(Color: TColor);
begin
  if Assigned(FControls.FColor) then
  begin
    if (FControls.FColor is TPanel) then TPanel(FControls.FColor).Color:=Color;
    if (FControls.FColor is TLabel) then TLabel(FControls.FColor).Color:=Color;
    if (FControls.FColor is TStaticText) then
TStaticText(FControls.FColor).Color:=Color;
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.OutXY(Fx,Fy: TFloat);
var
  Sx,Sy: Str32;
begin
  Sx:=FloatToStrF(Fx,ffFixed,7,3);
  Sy:=FloatToStrF(Fy,ffFixed,7,3);
```

```
 if Assigned(FControls.FXOut) then
 begin
  if (FControls.FXOut is TEdit) then TEdit(FControls.FXOut).Text:=Sx;
  if (FControls.FXOut is TPanel) then TPanel(FControls.FXOut).Caption:=Sx;
  if (TObject(FControls.FXOut) is TStatusPanel) then
TStatusPanel(FControls.FXOut).Text:=Sx;
  if (FControls.FXOut is TLabel) then TLabel(FControls.FXOut).Caption:=Sx;
  if (FControls.FXOut is TStaticText) then
TStaticText(FControls.FXOut).Caption:=Sx;
 end;

 if Assigned(FControls.FYOut) then
 begin
  if (FControls.FYOut is TEdit) then TEdit(FControls.FYOut).Text:=Sy;
  if (FControls.FYOut is TPanel) then TPanel(FControls.FYOut).Caption:=Sy;
  if (TObject(FControls.FYOut) is TStatusPanel) then
TStatusPanel(FControls.FYOut).Text:=Sy;
  if (FControls.FYOut is TLabel) then TLabel(FControls.FYOut).Caption:=Sy;
  if (FControls.FYOut is TStaticText) then
TStaticText(FControls.FYOut).Caption:=Sy;
 end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.OutAngle(A: TFloat);
var
 Sa: Str32;
begin
 if Assigned(FControls.FAngle) then
 begin
  if A > 9.9E16 then Sa:='' else Sa:=FloatToStrF(A,ffFixed,7,3);
  if (FControls.FAngle is TPanel) then TPanel(FControls.FAngle).Caption:=Sa;
  if (TObject(FControls.FAngle) is TStatusPanel) then
TStatusPanel(FControls.FAngle).Text:=Sa;
  if (FControls.FAngle is TLabel) then TLabel(FControls.FAngle).Caption:=Sa;
  if (FControls.FAngle is TStaticText) then
TStaticText(FControls.FAngle).Caption:=Sa;
 end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DoPan(Dx,Dy: Integer);
begin
 if not PanCurves then
 begin
  OutMode('Pan: Graph');
  SetEditEnable(False);
  if Dx <> 0 then FXAxis.Pan:=FXAxis.Pan + Dx;
  if Dy <> 0 then FYAxis.Pan:=FYAxis.Pan - Dy;
  Paint;
```

```
  end
  else
  begin
    OutMode('Pan: Curves');
    SetEditEnable(True);
    ChangeCurveOfs(Dx * FXAxis.FValuePerPixel,-Dy *
FYAxis.FValuePerPixel,True);
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DoZoom(Dx,Dy: Integer);
var
  AXDif,AYDif: TFloat;
  XDif,YDif: TFloat;
  ZFx,ZFy: TFloat;
  Factor: TFloat;
  XStep,YStep: TFloat;
begin
  AXDif:=FXAxis.FMax - FXAxis.FMin;
  AYDif:=FYAxis.FMax - FYAxis.FMin;
  XDif:=FXAxis.FMaxSave - FXAxis.FMinSave;
  YDif:=FYAxis.FMaxSave - FYAxis.FMinSave;
  ZFx:=1.0;
  ZFy:=1.0;
  if AXDif <> 0.0 then ZFx:=XDif / AXDif;
  if AYDif <> 0.0 then ZFy:=YDif / AYDif;

  XStep:=0.0;
  YStep:=0.0;

  if Dx > 0 then XStep:=-0.02 else if Dx < 0 then XStep:=0.02;
  if Dy > 0 then YStep:=-0.02 else if Dy < 0 then YStep:=0.02;

  if ZoomAspectRatio then
  begin
    OutMode('Zoom: Aspect');
    OutXY(ZFx,ZFx);
    Factor:=FXAxis.Zoom + XStep;
    if ((Factor < 1) and (ZFx < FMaxZoom)) or
      ((Factor > 1) and (ZFx > 1 / FMaxZoom)) then
    begin
      FXAxis.SetZoom(Factor);
      FYAxis.SetZoom(Factor);
      Paint;
    end;
  end
  else
  begin
    OutMode('Zoom: Free');
```

```
    OutXY(ZFx,ZFy);
    Factor:=FXAxis.Zoom + XStep;
    if ((Factor < 1) and (ZFx < FMaxZoom)) or
      ((Factor > 1) and (ZFx > 1 / FMaxZoom)) then FXAxis.SetZoom(Factor);
    Factor:=FXAxis.Zoom - YStep;
    if ((Factor < 1) and (ZFy < FMaxZoom)) or
      ((Factor > 1) and (ZFy > 1 / FMaxZoom)) then FYAxis.SetZoom(Factor);
    Paint;
  end;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DoMouse(X,Y: Integer);
begin
  if FMode = gmCursor then OutMode('Mouse: Cursor')
    else OutMode('Mouse: Position');
  OutXY(XAxis.Value(X - Positions.XAxisLeft),
       YAxis.Value(Height - Y - Positions.YAxisBottom));
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.SetMeasureCursor(X,Y: Integer);
begin
  ClearMarkBox;
  OutMode('Mouse: Cursor');
  OutXY(0,0);
  ClearMarkBox;
  CPMatch:=False;
  CPx:=XAxis.Value(X - Positions.XAxisLeft);
  CPy:=YAxis.Value(Height - Y - Positions.YAxisBottom);
  DrawMarkBox;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DoMeasureCursor(X,Y: Integer);
begin
  OutXY(XAxis.Value(X - Positions.XAxisLeft) - CPx,
       YAxis.Value(Height - Y - Positions.YAxisBottom) - CPy);
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DoMove(Dx,Dy: Integer);
begin
  if CPMatch then
  begin
    CPx:=CPx + Dx * FXAxis.ValuePerPixel;
    CPy:=CPy - Dy * FYAxis.ValuePerPixel;
    ChangePoint(CPCurve,CPIndex,CPx,CPy);
    OutXY(CPx,CPy);
    OutMode('Edit: Move');
```

```
    Paint;
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DoCheckCP(X,Y: Integer);
var
  Fx,Fy: TFloat;
begin
  CheckCurvePoints(X - Positions.XAxisLeft,Height - Y - Positions.YAxisBottom);
  if CPMatch then
  begin
    if ((LastCPCurve <> CPCurve) or (LastCPIndex <> CPIndex)) then
    begin
      OutXY(CPx,CPy);
      FCurve:=FCurveList.Items[CPCurve];
      OutCurve(FCurve.Name);
      OutItem(CPIndex);
      OutColor(FCurve.Color);
      LastCPCurve:=CPCurve;
      LastCPIndex:=CPIndex;
      if CPIndex < Pred(FCurve.FPoints.Count) then
      begin
        GetPoint(CPCurve,Succ(CPIndex),Fx,Fy);
        if Assigned(FControls.FAngle) then OutAngle(Angle(CPx,CPy,Fx,Fy));
      end;
    end;
  end
  else if not CPMatch then
  begin
    OutCurve('');
    OutItem(-1);
    OutColor(FColors.FGraphBkGnd);
    OutAngle(9.9E18);
    LastCPCurve:=-1;
    LastCPIndex:=-1;
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DblClicked(Sender: TObject);
begin
  Reset;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.MouseDown(Button: TMouseButton; Shift: TShiftState; X,Y:
Integer);
var
  Cx,Cy,Dx,Dy,Fx,Fy: TFloat;
```

```
  N: Integer;
begin
  inherited MouseDown(Button,Shift,X,Y);

  MouseX:=X;
  MouseY:=Y;

  if Button = mbLeft then
  begin
    if not CPMatch then DoPan(0,0) else DoMouse(X,Y);
    case FMode of
      gmMove: DoMove(0,0);
      gmInsert: if CPMatch then
                begin
                  GetPoint(CPCurve,CPIndex,CPx,CPy);
                  CPx:=XAxis.Value(X - Positions.XAxisLeft);
                  CPy:=YAxis.Value(Height - Y - Positions.YAxisBottom);
                  InsertPoint(CPCurve,CPIndex,CPx,CPy);
                  Paint;
                end;
      gmDelete: if CPMatch then
                begin
                  DeletePoint(CPCurve,CPIndex);
                  Paint;
                end;
      gmCursor: SetMeasureCursor(X,Y);
    end;
  end
  else if Button = mbRight then
  begin
    Cx:=FXAxis.GetCenter;
    Cy:=FYAxis.GetCenter;
    Dx:=FXAxis.Value(X - Positions.XAxisLeft);
    Dy:=FYAxis.Value(Height - Y - Positions.YAxisBottom);
    if Cx > Dx then Fx:=(Cx - Dx) / 10 else Fx:=(Dx - Cx) / 10;
    if Cy > Dy then Fy:=(Cy - Dy) / 10 else Fy:=(Dy - Cy) / 10;
    for N:=1 to 10 do
    begin
      if Cx > Dx then FXAxis.SetCenter(Cx - N * Fx)
        else FXAxis.SetCenter(Cx + N * Fx);
      if Cy > Dy then FYAxis.SetCenter(Cy - N * Fy)
        else FYAxis.SetCenter(Cy + N * Fy);
      Paint;
    end;
    DoZoom(0,0);
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.MouseMove(Shift: TShiftState; X,Y: Integer);
```

```
var
  Dx,Dy: Integer;
begin
  inherited MouseMove(Shift,X,Y);

  Dx:=X - MouseX;
  Dy:=Y - MouseY;

  Freeze:=ssShift in Shift;

  if ssLeft in Shift then
  begin
    if not CPMatch then DoPan(Dx,Dy);
    case FMode of
      gmMove: DoMove(Dx,Dy);
      gmCursor: DoMeasureCursor(X,Y);
    end;
  end
  else if ssRight in Shift then DoZoom(Dx,Dy)
  else
  begin
    if (FMode <> gmNone) and (FMode <> gmCursor) then DoCheckCP(X,Y);
    if FMode = gmCursor then DoMeasureCursor(X,Y)
    else if (FMode = gmNone) and (LastCPIndex > -1) then
DoCheckCP(MaxInt,MaxInt)
    else if not CPMatch then DoMouse(X,Y);
  end;

  if not PanCurves and (FMode = gmMove) then SetEditEnable(CPMatch)
    else if PanCurves then SetEditEnable(True);

  MouseX:=X;
  MouseY:=Y;
end;
{--------------------------------------------------------------------}

procedure TXYGraph.MouseUp(Button: TMouseButton; Shift: TShiftState; X,Y:
Integer);
begin
  inherited MouseUp(Button,Shift,X,Y);
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.SetGraphTitle(const Value: Str32);
begin
  if Value <> FGraphTitle then
  begin
    FGraphTitle:=Value;
    Paint;
  end;
```

```
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.Reset;
begin
  FXAxis.FZoom:=1.0;
  FYAxis.FZoom:=1.0;
  FXAxis.FPan:=0;
  FYAxis.FPan:=0;
  FXAxis.FMin:=FXAxis.FMinSave;
  FXAxis.FMax:=FXAxis.FMaxSave;
  FYAxis.FMin:=FYAxis.FMinSave;
  FYAxis.FMax:=FYAxis.FMaxSave;
  FXAxis.CalcAxis;
  FYAxis.CalcAxis;
  Zoom:=ZoomSave;
  Paint;
end;
{-------------------------------------------------------------------------------}

function TXYGraph.MakeCurve(const AName: Str32; AColor: TColor; ALineWidth:
Byte;
                    APenStyle: TPenStyle; AEnabled: Boolean): Integer;
var
  H,N: Integer;
  S: Str32;
begin
  N:=0;
  S:=AName;
  while GetCurveHandle(S,H) do
  begin
    Inc(N);
    S:=AName + IntToStr(N);
  end;
  FCurve:=TCurve.Create;
  FCurve.Name:=S;
  FCurve.Color:=AColor;
  FCurve.LineWidth:=ALineWidth;
  FCurve.PenStyle:=APenStyle;
  FCurve.Enabled:=AEnabled;
  FCurveList.Add(FCurve);
  Result:=FCurveList.IndexOf(FCurve);
  if Assigned(FControls.FViewListBox) then
  begin
    FControls.FViewListBox.Items.Add(S);

 FControls.FViewListBox.Checked[Pred(FControls.FViewListBox.Items.Count)]:=A
 Enabled;
  end;
  if Assigned(FControls.FPanListBox) then
```

```
  begin
    FControls.FPanListBox.Items.Add(S);

FControls.FPanListBox.Checked[Pred(FControls.FPanListBox.Items.Count)]:=False;
  end;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DeleteCurve(AItem: Integer);
begin
  if AItem < FCurveList.Count then
  begin
    FCurve:=FCurveList.Items[AItem];
    FCurveList.Delete(AItem);
    FCurve.Destroy;
  end;
end;
{--------------------------------------------------------------------------}

function TXYGraph.GetCurveHandle(AName: Str32; var H: Integer): Boolean;
var
  I,J: Integer;
begin
  H:=-1;
  J:=FCurveList.Count;
  I:=0;
  AName:=AnsiUpperCase(AName);
  while I < J do
  begin
    FCurve:=FCurveList.Items[I];
    if AnsiUpperCase(FCurve.Name) = AName then
    begin
      H:=I;
      Break;
    end;
    Inc(I);
  end;
  Result:=I < J;
end;
{--------------------------------------------------------------------------}

function TXYGraph.GetCurveName(H: Integer): Str32;
begin
  Result:='';
  if (H < 0) or (H > Pred(FCurveList.Count)) then Exit;
  FCurve:=FCurveList.Items[H];
  Result:=FCurve.Name;
end;
{--------------------------------------------------------------------------}
```

```
procedure TXYGraph.ChangePoint(AIndex,APosition: Integer; X,Y: TFloat);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  if APosition < FCurve.FPoints.Count then FCurve.ChangePoint(APosition,X,Y);
 end;
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.GetPoint(AIndex,APosition: Integer; var X,Y: TFloat);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  if InRange(APosition,0,Pred(FCurve.FPoints.Count)) then
   FCurve.GetPoint(APosition,X,Y);
 end;
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.AddPoint(AIndex: Integer; X,Y: TFloat);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  FCurve.AddPoint(X,Y);
 end;
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.AddMark(AIndex,APosition: Integer; AMarkType:
TMarkType; AColor: TColor);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  FCurve.AddMark(APosition,AMarkType,AColor);
 end;
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.SetMarkSize(AIndex: Integer; AMarkSize: TMarkSize);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  FCurve.FMarkSize:=AMarkSize;
 end;
end;
```

```
{-------------------------------------------------------------------------}

procedure TXYGraph.AddText(AIndex,APosition,AXOfs,AYOfs: Integer; const
AText: Str32; AColor: TColor);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  FCurve.AddText(APosition,AXOfs,AYOfs,AText,AColor);
 end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.SetCurveFont(AIndex: Integer; AName: TFontName; ASize:
Integer; AStyle: TFontStyles);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  FCurve.FFont.Name:=AName;
  FCurve.FFont.Size:=ASize;
  FCurve.FFont.Style:=AStyle;
 end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.InsertPoint(AIndex,APosition: Integer; X,Y: TFloat);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  FCurve.InsertPoint(APosition,X,Y);
 end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DeletePoint(AIndex,APosition: Integer);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
  FCurve:=FCurveList.Items[AIndex];
  FCurve.DeletePoint(APosition);
 end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.SetXOfs(AIndex: Integer; AOfs: TFloat);
begin
 if InRange(AIndex,0,Pred(FCurveList.Count)) then
 begin
```

```
    FCurve:=FCurveList.Items[AIndex];
    FCurve.XOfs:=AOfs;
    Paint;
   end;
end;
{--------------------------------------------------------------------------}

function TXYGraph.GetXOfs(AIndex: Integer): TFloat;
begin
  if InRange(AIndex,0,Pred(FCurveList.Count)) then
  begin
    FCurve:=FCurveList.Items[AIndex];
    Result:=FCurve.XOfs;
  end
  else Result:=0;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.SetYOfs(AIndex: Integer; AOfs: TFloat);
begin
  if InRange(AIndex,0,Pred(FCurveList.Count)) then
  begin
    FCurve:=FCurveList.Items[AIndex];
    FCurve.YOfs:=AOfs;
    Paint;
  end;
end;
{--------------------------------------------------------------------------}

function TXYGraph.GetYOfs(AIndex: Integer): TFloat;
begin
  if InRange(AIndex,0,Pred(FCurveList.Count)) then
  begin
    FCurve:=FCurveList.Items[AIndex];
    Result:=FCurve.YOfs;
  end
  else Result:=0;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.SetCurveEnabled(AIndex: Integer; Value: Boolean);
begin
  if InRange(AIndex,0,Pred(FCurveList.Count)) then
  begin
    FCurve:=FCurveList.Items[AIndex];
    FCurve.Enabled:=Value;
  end;
end;
{--------------------------------------------------------------------------}
```

```
procedure TXYGraph.SetZoom(Value: TFloat);
begin
 if (Value <= FMaxZoom) and (Value >= 1 / FMaxZoom) then
 begin
  FZoom:=Value;
  XAxis.SetZoom(FZoom);
  YAxis.SetZoom(FZoom);
  Paint;
 end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.ClearMarkBox;
begin
 if Assigned(CPBmp) then
 begin
  Canvas.CopyRect(CPRect,CPBmp.Canvas,Rect(0,0,8,8));
  CPBmp.Free;
  CPBmp:=nil;
 end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DrawMarkBox;
var
 I,J: Integer;
begin
 if Assigned(CPBmp) then ClearMarkBox;
 I:=FPositions.FXAxisLeft + FXAxis.Pixel(CPx);
 J:=Height - Positions.YAxisBottom - FYAxis.Pixel(CPy);
 CPRect:=Rect(I - 3,J - 3,I + 4,J + 4);
 CPBmp:=TBitMap.Create;
 CPBmp.Width:=8;
 CPBmp.Height:=8;
 CPBmp.Canvas.CopyRect(Rect(0,0,8,8),Canvas,CPRect);
 Canvas.Pen.Color:=clWhite;
 Canvas.Brush.Color:=clWhite;
 HClip:=CreateRectRgn(Positions.XAxisLeft,Positions.YAxisTop,
              Width - Positions.XAxisRight,Height - Positions.YAxisBottom);
 SelectClipRgn(Canvas.Handle,HClip);

 if FMode = gmCursor then
 begin
  Canvas.MoveTo(CPRect.Left + 1,CPRect.Top + 1);
  Canvas.LineTo(CPRect.Right,CPRect.Bottom);
  Canvas.MoveTo(CPRect.Right - 1,CPRect.Top + 1);
  Canvas.LineTo(CPRect.Left,CPRect.Bottom);
 end
 else Canvas.FrameRect(CPRect);
```

```
  DeleteObject(HClip);
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DrawMark(ACanvas: TCanvas; MarkType: TMarkType;
                    MarkColor: TColor; MarkSize: TMarkSize; X,Y: Integer);
begin
  ACanvas.Pen.Color:=MarkColor;
  ACanvas.Brush.Style:=bsClear;
  case MarkType of
    mtBox: begin
           ACanvas.MoveTo(X - MarkSize,Y - MarkSize);
           ACanvas.LineTo(X + MarkSize,Y - MarkSize);
           ACanvas.LineTo(X + MarkSize,Y + MarkSize);
           ACanvas.LineTo(X - MarkSize,Y + MarkSize);
           ACanvas.LineTo(X - MarkSize,Y - MarkSize);
         end;
    mtCircle:  ACanvas.Ellipse(X  -  MarkSize,Y  -  MarkSize,X  +  MarkSize  +  2,Y  +
 MarkSize + 2);
    mtCross: begin
             ACanvas.MoveTo(X - MarkSize + 1,Y - MarkSize + 1);
             ACanvas.LineTo(X + MarkSize,Y + MarkSize);
             ACanvas.MoveTo(X + MarkSize - 1,Y - MarkSize + 1);
             ACanvas.LineTo(X - MarkSize,Y + MarkSize);
           end;
  end;
 end;
 {-------------------------------------------------------------------------}

 procedure TXYGraph.CheckCurvePoints(X,Y: Integer);
 var
   I,J,K,L: Integer;
   Px,Py,Lx,Ly,Dx,Dy,MaxXDif,MaxYDif: TFloat;
 begin
   ClearMarkBox;
   if not Freeze or (Freeze and not CPMatch) then
   begin
     Px:=FXAxis.Value(X);
     Py:=FYAxis.Value(Y);

     MaxXDif:=10 * FXAxis.FValuePerPixel;
     MaxYDif:=10 * FYAxis.FValuePerPixel;

     CPMatch:=False;
     J:=Pred(FCurveList.Count);
     for I:=0 to J do
     begin
       FCurve:=FCurveList.Items[I];
       if FCurve.FEnabled then
       begin
```

```
      K:=Pred(FCurve.FPoints.Count);
      for L:=0 to K do
      begin
        GetPoint(I,L,Lx,Ly);
        Dx:=Abs(Px - Lx);
        Dy:=Abs(Py - Ly);
        if not CPMatch then
        begin
          CPMatch:=(Dx < MaxXDif) and (Dy < MaxYDif);
          if CPMatch then
          begin
            CPx:=Lx;
            CPy:=Ly;
            CPCurve:=I;
            CPIndex:=L;
          end;
        end
        else
        begin
          if (Dx < Abs(Px - CPx)) and (Dy < MaxYDif) or
             (Dy < Abs(Py - CPy)) and (Dx < MaxXDif) then
          begin
            CPx:=Lx;
            CPy:=Ly;
            CPCurve:=I;
            CPIndex:=L;
          end;
        end;
      end;
    end;
  end;
  if CPMatch then DrawMarkBox;
end;
{--------------------------------------------------------------------}

procedure TXYGraph.ShowHintPanel(Show: Boolean);
begin
  FHintPanel.Visible:=Show;
  if Assigned(FControls.FHintPanel) then FControls.FHintPanel.Checked:=Show;
end;
{--------------------------------------------------------------------}

procedure TXYGraph.ChangeCPx(Fx: TFloat);
begin
  if (FMode = gmMove) and CPMatch then
  begin
    GetPoint(CPCurve,CPIndex,CPx,CPy);
    ChangePoint(CPCurve,CPIndex,Fx,CPy);
    CPx:=Fx;
```

```
    OutXY(CPx,CPy);
    Paint;
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.ChangeCPy(Fy: TFloat);
begin
  if (FMode = gmMove) and CPMatch then
  begin
    GetPoint(CPCurve,CPIndex,CPx,CPy);
    ChangePoint(CPCurve,CPIndex,CPx,Fy);
    CPy:=Fy;
    OutXY(CPx,CPy);
    Paint;
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.ChangeCurveOfs(Ox,Oy: TFloat; Relative: Boolean);
var
  N: Integer;
begin
  if Assigned(FControls.FPanListBox) then
    for N:=0 to Pred(FControls.FPanListBox.Items.Count) do
  begin
    if FControls.FPanListBox.Checked[N] then
    begin
      FCurve:=FCurveList.Items[N];
      if FCurve.FEnabled then
      begin
        if Relative then
        begin
          if Ox > -9.99E15 then SetXOfs(N,GetXOfs(N) + Ox);
          if Oy > -9.99E15 then SetYOfs(N,GetYOfs(N) + Oy);
        end
        else
        begin
          if Ox > -9.99E15 then SetXOfs(N,Ox);
          if Oy > -9.99E15 then SetYOfs(N,Oy);
        end;
        OutXY(GetXOfs(N),GetYOfs(N));
      end;
    end;
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.GetCPInfo(var CPMatch: Boolean; var CPCurve,CPIndex:
Integer);
```

```
begin
  CPMatch:=CPMatch;
  CPCurve:=CPCurve;
  CPIndex:=CPIndex;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DoXEditExit(Sender: TObject);
var
  F: TFloat;
begin
  if not AtoF(TEdit(FControls.FXIn).Text,F) then Exit;
  ChangeCPx(F);
  ChangeCurveOfs(F,-9.9E16,False);
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DoYEditExit(Sender: TObject);
var
  F: TFloat;
begin
  if not AtoF(TEdit(FControls.FYIn).Text,F) then Exit;
  ChangeCPy(F);
  ChangeCurveOfs(-9.9E16,F,False);
end;
{--------------------------------------------------------------------------}

constructor THintPanel.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Graph:=TXYGraph(AOwner);
  FStrings:=TStringList.Create;
  FStrings.OnChange:=DoStringsChange;
  SetBounds(5,5,0,0);
  Cursor:=crHandPoint;
  Moving:=False;
  Start:=True;
end;
{--------------------------------------------------------------------------}

destructor THintPanel.Destroy;
begin
  FStrings.Free;
  inherited Destroy;
end;
{--------------------------------------------------------------------------}

procedure THintPanel.Loaded;
begin
  inherited Loaded;
```

```
  Canvas.Font.Name:='MS Sans Serif';
  Canvas.Font.Size:=8;
  Canvas.Font.Color:=clBlack;
end;
{--------------------------------------------------------------------------}

procedure THintPanel.NewBounds;
var
  H,I,J,L,N,W: Integer;
begin
  J:=0;
  N:=0;
  if FStrings.Count > 0 then
  begin
    for I:=0 to Pred(FStrings.Count) do
    begin
      L:=Canvas.TextWidth(FStrings.Strings[I]);
      if L > J then J:=L;
      if Length(FStrings.Strings[I]) > 0 then Inc(N);
    end;

    H:=14 * N + 4;
    W:=J + 4;

    if (N > 0) and (N <= MaxHintLines) then
    begin
      Width:=W + 4;
      Height:=H;
    end;
  end;
end;
{--------------------------------------------------------------------------}

procedure THintPanel.DoStringsChange(Sender: TObject);
begin
  NewBounds;
end;
{--------------------------------------------------------------------------}

procedure THintPanel.MouseDown(Button: TMouseButton; Shift: TShiftState; X,Y:
Integer);
begin
  inherited MouseDown(Button,Shift,X,Y);
  Anchors:=[];
  MouseX:=X;
  MouseY:=Y;
  Graph.ClearMarkBox;
  Graph.CPMatch:=False;
  Moving:=True;
end;
```

```
{-------------------------------------------------------------------------}

procedure THintPanel.MouseMove(Shift: TShiftState; X,Y: Integer);
var
  Dx,Dy: Integer;
begin
  inherited MouseMove(Shift,X,Y);
  if ssLeft in Shift then
  begin
    Dx:=X - MouseX;
    Dy:=Y - MouseY;
    SetBounds(Left + Dx,Top + Dy,Width,Height);
  end;
end;
{-------------------------------------------------------------------------}

procedure THintPanel.MouseUp(Button: TMouseButton; Shift: TShiftState; X,Y:
Integer);
begin
  inherited MouseUp(Button,Shift,X,Y);
  Anchors:=[akRight,akTop];
  Moving:=False;
end;
{-------------------------------------------------------------------------}

procedure THintPanel.Paint;
var
  I,L,N: Integer;
begin
  inherited Paint;

  if Start then Start:=False;

  if FStrings.Count > 0 then
  begin
    N:=0;
    for I:=0 to Pred(FStrings.Count) do
    begin
      L:=Length(FStrings.Strings[I]);
      if (L > 0) and (N <= MaxHintLines) then Inc(N);
    end;

    for I:=0 to Pred(N) do Canvas.TextOut(2,2 + I * 14,FStrings.Strings[I]);
  end;
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.SetMode(Value: TMode);
begin
  FMode:=Value;
```

```
 case FMode of
  gmNone: if Assigned(FControls.FModeNone) then
          FControls.FModeNone.Checked:=True;
  gmMove: if Assigned(FControls.FModeMove) then
          FControls.FModeMove.Checked:=True;
  gmInsert: if Assigned(FControls.FModeInsert) then
            FControls.FModeInsert.Checked:=True;
  gmDelete: if Assigned(FControls.FModeDelete) then
            FControls.FModeDelete.Checked:=True;
  gmCursor: if Assigned(FControls.FModeCursor) then
            FControls.FModeCursor.Checked:=True;
 end;
end;
{-------------------------------------------------------------------------------}
{
procedure TXYGraph.SetEditMode(Value: TEditMode);
begin
 FEditMode:=Value;
 if (FEditMode = emMove) or (FEditMode = emInsert) then
 begin
  if Assigned(FControls.XOut) then FControls.XOut.Enabled:=True;
  if Assigned(FControls.YOut) then FControls.YOut.Enabled:=True;
 end
 else
 begin
  if Assigned(FControls.XOut) then FControls.XOut.Enabled:=False;
  if Assigned(FControls.YOut) then FControls.YOut.Enabled:=False;
 end;
 case FEditMode of
  emNone: if Assigned(FControls.FEditModeNone) then
          FControls.FEditModeNone.Checked:=True;
  emMove: if Assigned(FControls.FEditModeMove) then
          FControls.FEditModeMove.Checked:=True;
  emInsert: if Assigned(FControls.FEditModeInsert) then
            FControls.FEditModeInsert.Checked:=True;
  emDelete: if Assigned(FControls.FEditModeDelete) then
            FControls.FEditModeDelete.Checked:=True;
 end;
 if FEditMode <> emNone then SetMode(gmNone);
end;
}
{-------------------------------------------------------------------------------}
(*
procedure TXYGraph.SetPanMode(Value: TPanMode);
begin
 FPanMode:=Value;
 case FPanMode of
  pmGraph: if Assigned(FControls.FPanModeGraph) then
           FControls.FPanModeGraph.Checked:=True;
  pmCurves: if Assigned(FControls.FPanModeCurves) then
```

```
          begin
            FControls.FPanModeCurves.Checked:=True;
            if Assigned(FControls.XOut) then FControls.XOut.Enabled:=True;
            if Assigned(FControls.YOut) then FControls.YOut.Enabled:=True;
          end;
  end;
end;
*)
{-------------------------------------------------------------------------------}

procedure TXYGraph.DoButtonClick(Sender: TObject);
begin
  if Sender = FControls.FClear then
  begin
    while FCurveList.Count > 0 do DeleteCurve(0);
    FHintPanel.FStrings.Clear;
    ShowHintPanel(False);
    GraphTitle:='Graph-Title';
    if Assigned(FControls.FViewListBox) then FControls.FViewListBox.Items.Clear;
    if Assigned(FControls.FPanListBox) then FControls.FPanListBox.Items.Clear;
    Reset;
    Application.ProcessMessages;
    Paint;
  end
  else if (Sender = FControls.FOpenView) and Assigned(FControls.FViewListBox)
then
  begin
    FControls.FViewListBox.BringToFront;
    FControls.FViewListBox.Visible:=not FControls.FViewListBox.Visible;
  end
  else if (Sender = FControls.FOpenPan) and Assigned(FControls.FPanListBox) then
  begin
    FControls.FPanListBox.BringToFront;
    FControls.FPanListBox.Visible:=not FControls.FPanListBox.Visible;
  end
  else if Sender = FControls.FReset then Reset;
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.DoRadioButtonClick(Sender: TObject);
begin
  if Sender = FControls.FModeNone then FMode:=gmNone
  else if Sender = FControls.FModeMove then FMode:=gmMove
  else if Sender = FControls.FModeInsert then FMode:=gmInsert
  else if Sender = FControls.FModeDelete then FMode:=gmDelete
  else if Sender = FControls.FModeCursor then FMode:=gmCursor;
  SetEditEnable(FMode = gmMove);
end;
{-------------------------------------------------------------------------------}
```

```
procedure TXYGraph.DoCheckBoxClick(Sender: TObject);
begin
  if Sender = FControls.FAspectRatio then
    ZoomAspectRatio:=FControls.FAspectRatio.Checked
  else if Sender = FControls.FMainGrid then
  begin
    FXAxis.ShowMainGrid:=FControls.FMainGrid.Checked;
    FYAxis.ShowMainGrid:=FControls.FMainGrid.Checked;
  end
  else if Sender = FControls.FSubGrid then
  begin
    FXAxis.ShowSubGrid:=FControls.FSubGrid.Checked;
    FYAxis.ShowSubGrid:=FControls.FSubGrid.Checked;
  end
  else if Sender = FControls.FHintPanel then
    FHintPanel.Visible:=FControls.FHintPanel.Checked;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DoListBoxClickCheck(Sender: TObject);
var
  LB: TCheckListBox;
  N: Integer;

begin
  if Sender = FControls.FViewListBox then
  begin
    LB:=FControls.FViewListBox;
    for N:=0 to Pred(LB.Items.Count) do
    begin
      FCurve:=FCurveList.Items[N];
      FCurve.Enabled:=LB.Checked[N];
    end;
    Paint;
  end
  else if Sender = FControls.FPanListBox then
  begin
    PanCurves:=False;
    N:=0;
    if Assigned(FControls.FPanListBox) then
      while not PanCurves and (N < FControls.FPanListBox.Items.Count) do
    begin
      PanCurves:=FControls.FPanListBox.Checked[N];
      Inc(N);
    end;
    if FMode <> gmMove then SetEditEnable(PanCurves);
  end;
end;
{--------------------------------------------------------------------------}
```

```
constructor TControls.Create(AGraph: TXYGraph);
begin
  inherited Create;
  Graph:=AGraph;
  FXOut:=nil;
  FYOut:=nil;
  FMode:=nil;
  FCurve:=nil;
  FItem:=nil;
  FColor:=nil;
  FAngle:=nil;
  FXIn:=nil;
  FYIn:=nil;
  FClear:=nil;
  FOpenView:=nil;
  FOpenPan:=nil;
  FReset:=nil;
  FModeNone:=nil;
  FModeMove:=nil;
  FModeInsert:=nil;
  FModeDelete:=nil;
  FModeCursor:=nil;
  FAspectRatio:=nil;
  FMainGrid:=nil;
  FSubGrid:=nil;
  FHintPanel:=nil;
  FViewListBox:=nil;
  FPanListBox:=nil;
end;
{--------------------------------------------------------------------------}

procedure TControls.SetControl(Index: Integer; Value: TControl);
begin
  case Index of
    0: FXOut:=Value;
    1: FYOut:=Value;
    2: FMode:=Value;
    3: FCurve:=Value;
    4: FItem:=Value;
    5: FColor:=Value;
    6: FAngle:=Value;
  end;
end;
{--------------------------------------------------------------------------}

procedure TControls.SetEdit(Index: Integer; Value: TEdit);
begin
  case Index of
    0: begin
         FXIn:=Value;
```

```
      if Assigned(FXIn) then FXIn.OnExit:=Graph.DoXEditExit;
     end;
   1: begin
      FYIn:=Value;
      if Assigned(FYIn) then FYIn.OnExit:=Graph.DoYEditExit;
     end;
 end;
end;
{-------------------------------------------------------------------------------}

procedure TControls.SetButton(Index: Integer; Value: TButton);
begin
 case Index of
   0: begin
      FClear:=Value;
      if Assigned(FClear) then FClear.OnClick:=Graph.DoButtonClick;
     end;
   1: begin
      begin
       FOpenView:=Value;
       if Assigned(FOpenView) then FOpenView.OnClick:=Graph.DoButtonClick;
      end;
     end;
   2: begin
      FOpenPan:=Value;
      if Assigned(FOpenPan) then FOpenPan.OnClick:=Graph.DoButtonClick;
     end;
   3: begin
      FReset:=Value;
      if Assigned(FReset) then FReset.OnClick:=Graph.DoButtonClick;
     end;
 end;
end;
{-------------------------------------------------------------------------------}

procedure TControls.SetRadioButton(Index: Integer; Value: TRadioButton);
begin
 case Index of
   0: begin
      FModeNone:=Value;
      if Assigned(FModeNone) then
       FModeNone.OnClick:=Graph.DoRadioButtonClick;
     end;
   1: begin
      FModeMove:=Value;
      if Assigned(FModeMove) then
       FModeMove.OnClick:=Graph.DoRadioButtonClick;
     end;
   2: begin
      FModeInsert:=Value;
```

```
      if Assigned(FModeInsert) then
        FModeInsert.OnClick:=Graph.DoRadioButtonClick;
      end;
   3: begin
      FModeDelete:=Value;
      if Assigned(FModeDelete) then
        FModeDelete.OnClick:=Graph.DoRadioButtonClick;
      end;
   4: begin
      FModeCursor:=Value;
      if Assigned(FModeCursor) then
        FModeCursor.OnClick:=Graph.DoRadioButtonClick;
      end;
  end;
end;
{-------------------------------------------------------------------------------}

procedure TControls.SetCheckBox(Index: Integer; Value: TCheckBox);
begin
  case Index of
    0: begin
       FAspectRatio:=Value;
       if Assigned(FAspectRatio) then
         FAspectRatio.OnClick:=Graph.DoCheckBoxClick;
       end;
    1: begin
       FMainGrid:=Value;
       if Assigned(FMainGrid) then
         FMainGrid.OnClick:=Graph.DoCheckBoxClick;
       end;
    2: begin
       FSubGrid:=Value;
       if Assigned(FSubGrid) then
         FSubGrid.OnClick:=Graph.DoCheckBoxClick;
       end;
    3: begin
       FHintPanel:=Value;
       if Assigned(FHintPanel) then
         FHintPanel.OnClick:=Graph.DoCheckBoxClick;
       end;
  end;
end;
{-------------------------------------------------------------------------------}

procedure TControls.SetListBox(Index: Integer; Value: TCheckListBox);
begin
  case Index of
    0: begin
       FViewListBox:=Value;
       if Assigned(FViewListBox) then
```

```
        FViewListBox.OnClickCheck:=Graph.DoListBoxClickCheck;
      end;
    1: begin
        FPanListBox:=Value;
        if Assigned(FPanListBox) then
         FPanListBox.OnClickCheck:=Graph.DoListBoxClickCheck;
      end;
  end;
end;
{--------------------------------------------------------------------------}

function TXYGraph.SaveCurveToStream(FileStream: TFileStream; Item: Integer):
Boolean;
var
  CurveData: TCurveData;
  N: Integer;
begin
  Result:=False;
  if not InRange(Item,0,Pred(FCurveList.Count)) or not Assigned(FileStream) then
Exit;
  FCurve:=FCurveList.Items[Item];
  try
    CurveData.Name:=FCurve.Name;
    CurveData.Enabled:=FCurve.Enabled;
    CurveData.Color:=FCurve.Color;
    CurveData.LineWidth:=FCurve.LineWidth;
    CurveData.PenStyle:=FCurve.PenStyle;
    CurveData.Points:=FCurve.FPoints.Count;
    CurveData.Texts:=FCurve.FTexts.Count;
    CurveData.Marks:=FCurve.FMarks.Count;
    CurveData.XOfs:=FCurve.XOfs;
    CurveData.YOfs:=FCurve.YOfs;
    CurveData.FontName:=FCurve.FFont.Name;
    CurveData.FontSize:=FCurve.FFont.Size;
    CurveData.FontStyle:=FCurve.FFont.Style;
    CurveData.MarkSize:=FCurve.MarkSize;

    FileStream.Write(CurveData,SizeOf(TCurveData));

    for N:=0 to Pred(FCurve.FPoints.Count) do
      FileStream.Write(FCurve.FPoints.Items[N]^,SizeOf(TPointRec));

    for N:=0 to Pred(FCurve.FTexts.Count) do
      FileStream.Write(FCurve.FTexts.Items[N]^,SizeOf(TTextRec));

    for N:=0 to Pred(FCurve.FMarks.Count) do
      FileStream.Write(FCurve.FMarks.Items[N]^,SizeOf(TMarkRec));

    Result:=True;
  except
```

```
    ShowMessage('Error writing stream!');
  end;
end;
{--------------------------------------------------------------------------}

function TXYGraph.LoadCurveFromStream(FileStream: TFileStream): Boolean;
var
  CurveData: TCurveData;
  PointRec: TPointRec;
  TextRec: TTextRec;
  MarkRec: TMarkRec;
  H,N: Integer;
begin
  Result:=False;
  if not Assigned(FileStream) then Exit;
  try
    FileStream.Read(CurveData,SizeOf(TCurveData));
    H:=MakeCurve(CurveData.Name,CurveData.Color,CurveData.LineWidth,
              CurveData.PenStyle,CurveData.Enabled);
    SetXOfs(H,CurveData.XOfs);
    SetYOfs(H,CurveData.YOfs);
    SetCurveFont(H,CurveData.FontName,CurveData.FontSize,CurveData.FontStyle);
    SetMarkSize(H,CurveData.MarkSize);

    for N:=0 to Pred(CurveData.Points) do
    begin
      FileStream.Read(PointRec,SizeOf(TPointRec));
      AddPoint(H,PointRec.x,PointRec.y);
    end;

    for N:=0 to Pred(CurveData.Texts) do
    begin
      FileStream.Read(TextRec,SizeOf(TTextRec));
      AddText(H,TextRec.PointIndex,TextRec.XOfs,TextRec.YOfs,
            TextRec.Text,TextRec.TextColor);
    end;

    for N:=0 to Pred(CurveData.Marks) do
    begin
      FileStream.Read(MarkRec,SizeOf(TMarkRec));
      AddMark(H,MarkRec.PointIndex,MarkRec.MarkType,MarkRec.MarkColor);
    end;

    Result:=True;
  except
    ShowMessage('Error reading stream!');
  end;
end;
{--------------------------------------------------------------------------}
```

```
function TXYGraph.SaveCurveToFile(const FileName: string; Item: Integer):
Boolean;
var
  FileStream: TFileStream;
begin
  Result:=False;
  try
    FileStream:=TFileStream.Create(FileName,fmCreate);
    try
      FileStream.Position:=0;
      Result:=SaveCurveToStream(FileStream,Item);
    except
      Result:=False;
    end;
  finally
    FileStream.Free;
  end;
end;
{--------------------------------------------------------------------------}

function TXYGraph.LoadCurveFromFile(const FileName: string): Boolean;
var
  FileStream: TFileStream;
begin
  Result:=False;
  if not FileExists(FileName) then Exit;
  try
    FileStream:=TFileStream.Create(FileName,fmOpenRead);
    try
      FileStream.Position:=0;
      Result:=LoadCurveFromStream(FileStream);
    except
      Result:=False;
    end;
  finally
    FileStream.Free;
  end;
end;
{--------------------------------------------------------------------------}

function TXYGraph.SaveGraphToFile(const FileName: string): Boolean;
var
  FileStream: TFileStream;
  GraphData: TGraphData;
  FontRec: TFontRec;
  N: Integer;
begin
  Result:=False;
  try
    FileStream:=TFileStream.Create(FileName,fmCreate);
```

```
    try
      GraphData.GraphTitle:=FGraphTitle;
      GraphData.Curves:=FCurveList.Count;
      GraphData.Zoom:=FZoom;
      GraphData.MaxZoom:=FMaxZoom;
      FileStream.Position:=0;
      FileStream.Write(GraphData,SizeOf(GraphData));
      FileStream.Write(FXAxis.FTitle,FXAxis.InstanceSize);
      FileStream.Write(FYAxis.FTitle,FYAxis.InstanceSize);
      FileStream.Write(FColors.FAxisBkGnd,FColors.InstanceSize);
      FileStream.Write(FPositions.FXAxisLeft,FPositions.InstanceSize);

      FontRec.AxisScaleFontName:=FFonts.FAxisScale.Name;
      FontRec.AxisScaleFontSize:=FFonts.FAxisScale.Size;
      FontRec.AxisScaleFontStyle:=FFonts.FAxisScale.Style;
      FontRec.AxisTitleFontName:=FFonts.FAxisTitle.Name;
      FontRec.AxisTitleFontSize:=FFonts.FAxisTitle.Size;
      FontRec.AxisTitleFontStyle:=FFonts.FAxisTitle.Style;
      FontRec.GraphTitleFontName:=FFonts.FGraphTitle.Name;
      FontRec.GraphTitleFontSize:=FFonts.FGraphTitle.Size;
      FontRec.GraphTitleFontStyle:=FFonts.FGraphTitle.Style;
      FileStream.Write(FontRec,SizeOf(FontRec));

      for N:=0 to Pred(GraphData.Curves) do SaveCurveToStream(FileStream,N);
      FHintPanel.FStrings.SaveToStream(FileStream);
      Result:=True;
    except
      Result:=False;
    end;
  finally
    FileStream.Free;
  end;
end;
{-------------------------------------------------------------------------}

function TXYGraph.LoadGraphFromFile(const FileName: string): Boolean;
var
  FileStream: TFileStream;
  GraphData: TGraphData;
  FontRec: TFontRec;
  N: Integer;
begin
  Result:=False;
  if not FileExists(FileName) then Exit;
  try
    FileStream:=TFileStream.Create(FileName,fmOpenRead);
    try
      FileStream.Position:=0;
      FileStream.Read(GraphData,SizeOf(GraphData));
      FGraphTitle:=GraphData.GraphTitle;
```

```
    FZoom:=GraphData.Zoom;
    FMaxZoom:=GraphData.MaxZoom;
    FileStream.Read(FXAxis.FTitle,FXAxis.InstanceSize);
    FileStream.Read(FYAxis.FTitle,FYAxis.InstanceSize);
    FileStream.Read(FColors.FAxisBkGnd,FColors.InstanceSize);
    FileStream.Read(FPositions.FXAxisLeft,FPositions.InstanceSize);

    FileStream.Read(FontRec,SizeOf(FontRec));
    FFonts.FAxisScale.Name:=FontRec.AxisScaleFontName;
    FFonts.FAxisScale.Size:=FontRec.AxisScaleFontSize;
    FFonts.FAxisScale.Style:=FontRec.AxisScaleFontStyle;
    FFonts.FAxisTitle.Name:=FontRec.AxisTitleFontName;
    FFonts.FAxisTitle.Size:=FontRec.AxisTitleFontSize;
    FFonts.FAxisTitle.Style:=FontRec.AxisTitleFontStyle;
    FFonts.FGraphTitle.Name:=FontRec.GraphTitleFontName;
    FFonts.FGraphTitle.Size:=FontRec.GraphTitleFontSize;
    FFonts.FGraphTitle.Style:=FontRec.GraphTitleFontStyle;

    for N:=0 to Pred(GraphData.Curves) do LoadCurveFromStream(FileStream);
    FHintPanel.FStrings.LoadFromStream(FileStream);
    FXAxis.CalcAxis;
    FYAxis.CalcAxis;
    FHintPanel.Paint;
    Paint;
    Result:=True;
   except
    Result:=False;
   end;
  finally
   FileStream.Free;
  end;
end;
{--------------------------------------------------------------------------}

procedure TXYGraph.DXFAxis;
var
  Dif,MainStep,SubStep: TFloat;
  TickLen,TextWidth: TFloat;
  N,M: Integer;
  S: Str32;
begin
  if not Assigned(DXFOut) then Exit;
  TextWidth:=DXFOut.TextHeight / 4 * 3;
  TickLen:=5;
  Dif:=FXAxis.FMax - FXAxis.FMin;
  if FXAxis.MainTicks <> 0 then MainStep:=Dif / FXAxis.MainTicks else
MainStep:=Dif;
  if FXAxis.SubTicks <> 0 then SubStep:=MainStep / FXAxis.SubTicks else
SubStep:=MainStep;
  for N:=0 to FXAxis.MainTicks do
```

```
 begin
  DXFOut.Line(DXFOut.ToX(FXAxis.FMin + N * MainStep),
          DXFOut.ToY(FYAxis.FMin) - TickLen,0,
          DXFOut.ToX(FXAxis.FMin + N * MainStep),
          DXFOut.ToY(FYAxis.FMax),0);
  S:=FloatToStrF(FXAxis.Min + (N * FXAxis.ValuePerMainTick),
           ffFixed,7,FXAxis.Decimals);
  for M:=1 to Length(S) do if S[M] = ' ' then Delete(S,M,1);
  DXFOut.DText(DXFOut.ToX(FXAxis.FMin + N * MainStep) - TextWidth *
Length(S) / 2,
          DXFOut.ToY(FYAxis.FMin) - DXFOut.TextHeight - TickLen - 1,0,
          DXFOut.TextHeight,0,S);
  if FXAxis.FShowSubGrid and (N < FXAxis.MainTicks) then
   for M:=1 to Pred(FXAxis.SubTicks) do
  begin
   DXFOut.Line(DXFOut.ToX(FXAxis.FMin + N * MainStep + M * SubStep),
           DXFOut.ToY(FYAxis.FMin),0,
           DXFOut.ToX(FXAxis.FMin + N * MainStep + M * SubStep),
           DXFOut.ToY(FYAxis.FMax),0);
  end;
 end;

 Dif:=FYAxis.FMax - FYAxis.FMin;
 if FYAxis.MainTicks <> 0 then MainStep:=Dif / FYAxis.MainTicks else
MainStep:=Dif;
 if FYAxis.SubTicks <> 0 then SubStep:=MainStep / FYAxis.SubTicks else
SubStep:=MainStep;
 for N:=0 to FYAxis.MainTicks do
 begin
  DXFOut.Line(DXFOut.ToX(FXAxis.FMin) - TickLen,
          DXFOut.ToY(FYAxis.FMin + N * MainStep),0,
          DXFOut.ToX(FXAxis.FMax),
          DXFOut.ToY(FYAxis.FMin + N * MainStep),0);
  S:=FloatToStrF(FYAxis.Min + (N * FYAxis.ValuePerMainTick),
           ffFixed,7,FYAxis.Decimals);
  for M:=1 to Length(S) do if S[M] = ' ' then Delete(S,M,1);
  DXFOut.DText(DXFOut.ToX(FXAxis.FMin) - TickLen - TextWidth * Length(S) -
1,
          DXFOut.ToY(FYAxis.FMin + N * MainStep) - DXFOut.TextHeight / 2,
          0,DXFOut.TextHeight,0,S);
  if FXAxis.FShowSubGrid and (N < FYAxis.MainTicks) then
   for M:=1 to Pred(FYAxis.SubTicks) do
  begin
   DXFOut.Line(DXFOut.ToX(FXAxis.FMin),
           DXFOut.ToY(FYAxis.FMin + N * MainStep + M * SubStep),0,
           DXFOut.ToX(FXAxis.FMax),
           DXFOut.ToY(FYAxis.FMin + N * MainStep + M * SubStep),0);
  end;
 end;
```

```
  S:=FXAxis.FTitle;
  SubStep:=Length(S) * TextWidth;
  MainStep:=(FXAxis.FMax - FXAxis.FMin) / 2;
  Dif:=FXAxis.FMin + MainStep - SubStep;
  DXFOut.DText(DXFOut.ToX(Dif),
          DXFOut.ToY(FYAxis.FMin) - TickLen - DXFOut.TextHeight * 6,
          0,DXFOut.TextHeight * 2,0,S);

  S:=FYAxis.FTitle;
  SubStep:=Length(S) * TextWidth;
  MainStep:=(FYAxis.FMax - FYAxis.FMin) / 2;
  Dif:=FYAxis.FMin + MainStep - SubStep;
  DXFOut.DText(DXFOut.ToX(FXAxis.FMin) - TickLen - TextWidth * Length(S) -
          DXFOut.TextHeight * 2 - 2,
          DXFOut.ToY(Dif),0,DXFOut.TextHeight * 2,90,S);

  S:=FGraphTitle;
  SubStep:=Length(S) * TextWidth;
  MainStep:=(FXAxis.FMax - FXAxis.FMin) / 2;
  Dif:=FXAxis.FMin + MainStep - SubStep;
  DXFOut.DText(DXFOut.ToX(Dif),
          DXFOut.ToY(FYAxis.FMax) + DXFOut.TextHeight * 6,
          0,DXFOut.TextHeight * 2,0,S);
end;
{-------------------------------------------------------------------------}

procedure TXYGraph.DXFCurves;
var
  H,I,J: Integer;
  X,Y: TFloat;
begin
  for H:=0 to Pred(FCurveList.Count) do
  begin
    FCurve:=FCurveList.Items[H];
    if FCurve.Enabled and (FCurve.FPoints.Count > 0) then
    begin
      J:=Pred(FCurve.FPoints.Count);
      DXFOut.StartPolyLine(False);
      for I:=0 to J do
      begin
        FCurve.GetPoint(I,X,Y);
        DXFOut.Vertex(DXFOut.ToX(X),DXFOut.ToY(Y),0);
      end;
      DXFOut.EndPolyLine;
    end;
  end;
end;
{-------------------------------------------------------------------------}
```

```
function TXYGraph.MakeDXF(const FileName: string;
FromX1,FromY1,FromX2,FromY2,
                ToX1,ToY1,ToX2,ToY2,TextHeight: TFloat;
                Decimals: Byte): Boolean;
begin
  Result:=False;
  try

DXFOut:=TDXFOut.Create(FromX1,FromY1,FromX2,FromY2,ToX1,ToY1,ToX2,T
oY2,
                TextHeight,Decimals);
    try
      DXFOut.Header;
      DXFAxis;
      DXFCurves;
      DXFOut.Trailer;
      DXFOut.StringList.SaveToFile(FileName);
      Result:=True;
    except
      Result:=False;
    end;
  finally
    DXFOut.Free;
  end;
end;
{-------------------------------------------------------------------------}

constructor TDXFOut.Create(AFromXMin,AFromYMin,AFromXMax,AFromYMax,
                ATOXMin,ATOYMin,ATOXMax,ATOYMax,ATextHeight: TFloat;
ADecimals: Byte);
begin
  inherited Create;
  FromXMin:=AFromXMin;
  FromYMin:=AFromYMin;
  FromXMax:=AFromXMax;
  FromYMax:=AFromYMax;
  ToXMin:=AToXMin;
  ToYMin:=AToYMin;
  ToXMax:=AToXMax;
  ToYMax:=AToYMax;
  TextHeight:=ATextHeight;
  Decimals:=ADecimals;
  StringList:=TStringList.Create;
end;
{-------------------------------------------------------------------------}

destructor TDXFOut.Destroy;
begin
  StringList.Free;
  inherited Destroy;
```

```
end;
{-------------------------------------------------------------------------------}

procedure TDXFOut.Header;
begin
  LayerName:='0';
  StringList.Add('0');
  StringList.Add('SECTION');
  StringList.Add('2');
  StringList.Add('HEADER');
  StringList.Add('9');
  StringList.Add('$LIMMIN');
  StringList.Add('10');
  StringList.Add(FToA(ToXMin));
  StringList.Add('20');
  StringList.Add(FToA(ToYMin));
  StringList.Add('9');
  StringList.Add('$LIMMAX');
  StringList.Add('10');
  StringList.Add(FToA(ToXMax));
  StringList.Add('20');
  StringList.Add(FToA(ToYMax));
  StringList.Add('0');
  StringList.Add('ENDSEC');
  StringList.Add('0');
  StringList.Add('SECTION');
  StringList.Add('2');
  StringList.Add('TABLES');
  StringList.Add('0');
  StringList.Add('TABLE');
  StringList.Add('2');
  StringList.Add('LAYER');
  StringList.Add('70');
  StringList.Add('1');
  StringList.Add('0');
  StringList.Add('LAYER');
  StringList.Add('2');
  StringList.Add('0');
  StringList.Add('70');
  StringList.Add('64');
  StringList.Add('62');
  StringList.Add('7');
  StringList.Add('6');
  StringList.Add('CONTINUOUS');
  StringList.Add('0');
  StringList.Add('ENDTAB');
  StringList.Add('0');
  StringList.Add('ENDSEC');
  StringList.Add('0');
  StringList.Add('SECTION');
```

```
  StringList.Add('2');
  StringList.Add('ENTITIES');
end;
{-------------------------------------------------------------------------}

function TDXFOut.FToA(F: TFloat): Str32;
var
  I: Integer;
begin
  Result:=FloatToStrF(F,ffFixed,16,Decimals);
  I:=Pos(',',Result);
  if I > 0 then Result[I]:='.';
end;
{-------------------------------------------------------------------------}

function TDXFOut.ToX(X: TFloat): TFloat;
var
  Factor,FromDif: TFloat;
begin
  FromDif:=FromXMax - FromXMin;
  if FromDif <> 0.0 then Factor:=(ToXMax - ToXMin) / FromDif else Factor:=1.0;
  Result:=X * Factor;
end;
{-------------------------------------------------------------------------}

function TDXFOut.ToY(Y: TFloat): TFloat;
var
  Factor,FromDif: TFloat;
begin
  FromDif:=FromYMax - FromYMin;
  if FromDif <> 0.0 then Factor:=(ToYMax - ToYMin) / FromDif else Factor:=1.0;
  Result:=Y * Factor;
end;
{-------------------------------------------------------------------------}

procedure TDXFOut.SetLayer(const Name: Str32);
begin
  LayerName:=Name;
end;
{-------------------------------------------------------------------------}

procedure TDXFOut.Layer;
begin
  StringList.Add('8');
  StringList.Add(LayerName);
end;
{-------------------------------------------------------------------------}

procedure TDXFOut.StartPoint(X,Y,Z: TFloat);
begin
```

```
    StringList.Add('10');
    StringList.Add(FToA(X));
    StringList.Add('20');
    StringList.Add(FToA(Y));
    StringList.Add('30');
    StringList.Add(FToA(Z));
  end;
  {-------------------------------------------------------------------------}

  procedure TDXFOut.EndPoint(X,Y,Z: TFloat);
  begin
    StringList.Add('11');
    StringList.Add(FToA(X));
    StringList.Add('21');
    StringList.Add(FToA(Y));
    StringList.Add('31');
    StringList.Add(FToA(Z));
  end;
  {-------------------------------------------------------------------------}

  procedure TDXFOut.AddText(const Txt: Str32);
  begin
    StringList.Add('1');
    StringList.Add(Txt);
  end;
  {-------------------------------------------------------------------------}

  procedure TDXFOut.StartPolyLine(Closed: Boolean);
  var
    Flag : Byte;
  begin
    StringList.Add('0');
    StringList.Add('POLYLINE');
    Layer;
    StringList.Add('66');
    StringList.Add('1');
    StartPoint(0,0,0);
    Flag:=8;
    if Closed then Flag:=Flag or 1;
    StringList.Add('70');
    StringList.Add(IntToStr(Flag));
  end;
  {-------------------------------------------------------------------------}

  procedure TDXFOut.Vertex(X,Y,Z: TFloat);
  var
   Flag : Byte;
  begin
    StringList.Add('0');
    StringList.Add('VERTEX');
```

```
  Layer;
  StartPoint(X,Y,Z);
  StringList.Add('70');
  Flag:=32;
  StringList.Add(IntToStr(Flag));
end;
{-------------------------------------------------------------------------}

procedure TDXFOut.EndPolyLine;
begin
  StringList.Add('0');
  StringList.Add('SEQEND');
  Layer;
end;
{-------------------------------------------------------------------------}

procedure TDXFOut.Line(X1,Y1,Z1,X2,Y2,Z2: TFloat);
begin
  StringList.Add('0');
  StringList.Add('LINE');
  Layer;
  StartPoint(X1,Y1,Z1);
  EndPoint(X2,Y2,Z2);
end;
{-------------------------------------------------------------------------}

procedure TDXFOut.Point(X,Y,Z: TFloat);
begin
  StringList.Add('0');
  StringList.Add('POINT');
  Layer;
  StartPoint(X,Y,Z);
end;
{-------------------------------------------------------------------------}

procedure TDXFOut.DText(X,Y,Z,Height,Angle: TFloat; const Txt: Str32);
begin
  StringList.Add('0');
  StringList.Add('TEXT');
  Layer;
  StartPoint(X,Y,Z);
  StringList.Add('40');
  StringList.Add(FToA(Height));
  AddText(Txt);
  StringList.Add('50');
  StringList.Add(FToA(Angle));
end;
{-------------------------------------------------------------------------}

procedure TDXFOut.Trailer;
```

```
begin
  StringList.Add('0');
  StringList.Add('ENDSEC');
  StringList.Add('0');
  StringList.Add('EOF');
end;
{-------------------------------------------------------------------------------}

procedure TXYGraph.Notification(Component: TComponent; Operation:
TOperation);
begin
  inherited Notification(Component,Operation);

  if Operation = opRemove then
  begin
    if Component = FControls.FXOut then FControls.FXOut:=nil;
    if Component = FControls.FYOut then FControls.FYOut:=nil;
    if Component = FControls.FMode then FControls.FMode:=nil;
    if Component = FControls.FCurve then FControls.FCurve:=nil;
    if Component = FControls.FItem then FControls.FItem:=nil;
    if Component = FControls.FColor then FControls.FColor:=nil;
    if Component = FControls.FAngle then FControls.FAngle:=nil;
    if Component = FControls.FXIn then FControls.FXIn:=nil;
    if Component = FControls.FYIn then FControls.FYIn:=nil;
    if Component = FControls.FClear then FControls.FClear:=nil;
    if Component = FControls.FOpenView then FControls.FOpenView:=nil;
    if Component = FControls.FOpenPan then FControls.FOpenPan:=nil;
    if Component = FControls.FReset then FControls.FReset:=nil;
    if Component = FControls.FModeNone then FControls.FModeNone:=nil;
    if Component = FControls.FModeMove then FControls.FModeMove:=nil;
    if Component = FControls.FModeInsert then FControls.FModeInsert:=nil;
    if Component = FControls.FModeDelete then FControls.FModeDelete:=nil;
    if Component = FControls.FModeCursor then FControls.FModeCursor:=nil;
    if Component = FControls.FAspectRatio then FControls.FAspectRatio:=nil;
    if Component = FControls.FMainGrid then FControls.FMainGrid:=nil;
    if Component = FControls.FSubGrid then FControls.FSubGrid:=nil;
    if Component = FControls.FHintPanel then FControls.FHintPanel:=nil;
    if Component = FControls.FViewListBox then FControls.FViewListBox:=nil;
    if Component = FControls.FPanListBox then FControls.FPanListBox:=nil;
  end;
end;
{-------------------------------------------------------------------------------}

initialization
end.
```

```
unit DelayEQ;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;
const ArraySize = 60; { Maximum number of equations }
type complex = record
    Re, Im: double;
  end;
  CompVector = array[0..ArraySize] of complex;
  IntVector = array[0..ArraySize] of integer;
  RealVector = array[1..ArraySize] of double;
  matrix = array[1..ArraySize] of RealVector;
function DifF(x: double): double;
procedure MinMax(LeftEndPoint: double;
  RightEndPoint: double;
  var Answer: double;
  var Error: byte);
procedure Partial_Pivoting(Dimen: integer;
  Coefficients: matrix;
  Constants: RealVector;
  var Solution: RealVector;
  var Error: byte);


implementation


function DifF(x: double): double;
const Tolerance = 1E-10;
type vector = array[1..100] of double;
var Term, Iter, TwoToTheIterMinus2, Extrap: integer;
  DeltaX, FourToTheExtrapMinus1: double;
  OldEstimate, NewEstimate: vector;

  function EvaluateFirstDeriv(X: double;
    DeltaX: double): double;
  var LeftPoint, RightPoint: double;
  begin
    LeftPoint := EquF(X - DeltaX);
    RightPoint := EquF(X + DeltaX);
    EvaluateFirstDeriv := (RightPoint - LeftPoint) / (2 * DeltaX);
  end;

begin
  if ABS(X) < Tolerance then
    DeltaX := Sqrt(Tolerance)
  else
    DeltaX := ABS(X * Sqrt(Tolerance));
```

```
  OldEstimate[1] := EvaluateFirstDeriv(X, DeltaX);
  Iter := 1;
  TwoToTheIterMinus2 := 1;
  repeat
    Iter := Succ(Iter);
    DeltaX := DeltaX / 2;
    NewEstimate[1] := EvaluateFirstDeriv(X, DeltaX);
    TwoToTheIterMinus2 := TwoToTheIterMinus2 * 2;
    FourToTheExtrapMinus1 := 1;
    for Extrap := 2 to Iter do
    begin
      FourToTheExtrapMinus1 := FourToTheExtrapMinus1 * 4;
      NewEstimate[Extrap] :=
        (FourToTheExtrapMinus1 * NewEstimate[Extrap - 1]
        - OldEstimate[Extrap - 1]) / (FourToTheExtrapMinus1 - 1);
    end;
    OldEstimate := NewEstimate;
  until (ABS(NewEstimate[Iter - 1] - NewEstimate[Iter]) <=
    ABS(Tolerance * NewEstimate[Iter])) or
    (ABS(DeltaX) < Tolerance);
  DiffF := NewEstimate[Iter];
end;


procedure MinMax(LeftEndPoint: double;
  RightEndPoint: double;
  var Answer: double;
  var Error: byte);
const NearlyZero = 1E-10;
var Tol, yLeft, yRight, MidPoint, yMidPoint: double;
  Iter, MaxIter: integer;
  Found: boolean;

  function TestForRoot(X, OldX, Y, Tol: double): boolean;
  begin
    TestForRoot := (ABS(Y) <= NearlyZero) or (ABS(X - OldX) < ABS(OldX * Tol));
  end;

begin
  Maxiter := 500;
  Tol := 1E-10;
  Error := 0;
  Found := false;
  yLeft := DiffF(LeftEndpoint);
  yRight := DiffF(RightEndpoint);
  if ABS(yLeft) <= NearlyZero then
  begin
    Answer := LeftEndpoint;
    Found := true;
  end;
```

```
  if ABS(yRight) <= NearlyZero then
  begin
    Answer := RightEndpoint;
    Found := true;
  end;
  if not Found then
  begin
    if yLeft * yRight > 0 then
      Error := 2;
    if Tol <= 0 then
      Error := 3;
    if MaxIter < 0 then
      Error := 4;
  end;
  if (Error = 0) and (Found = false) then
  begin
    Iter := 0;
    yLeft := DifF(LeftEndPoint);
    while not (Found) and (Iter < MaxIter) do
    begin
      Iter := Succ(Iter);
      MidPoint := (LeftEndpoint + RightEndpoint) / 2;
      yMidPoint := DifF(MidPoint);
      Found := TestForRoot(MidPoint, LeftEndpoint, yMidPoint, Tol);
      if (yLeft * yMidPoint) < 0 then
        RightEndpoint := MidPoint
      else
      begin
        LeftEndpoint := MidPoint;
        yLeft := yMidPoint;
      end;
    end;
    Answer := MidPoint;
    if Iter >= MaxIter then
      Error := 1;
  end;
end;

procedure Partial_Pivoting(Dimen: integer;
  Coefficients: matrix;
  Constants: RealVector;
  var Solution: RealVector;
  var Error: byte);
const NearlyZero = 1E-10;
var Sum, Multiplier, Dummy: double;
  Term, Row, ReferenceRow, PivotRow: integer;
  DummyRow: RealVector;

  procedure EROmultAdd(Multiplier: double;
    Dimen: integer;
```

```
    var ReferenceRow: RealVector;
    var ChangingRow: RealVector);
  var Term: integer;
  begin
    for Term := 1 to Dimen do
      ChangingRow[Term] := ChangingRow[Term] +
        Multiplier * ReferenceRow[Term];
  end;

begin
  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
        { Make Coefficients matrix upper triangular }
    ReferenceRow := 0;
    while (Error = 0) and (ReferenceRow < Dimen - 1) do
    begin
      ReferenceRow := Succ(ReferenceRow);
        { Find row with largest element in this column }
        { and switch this row with the ReferenceRow    }
      PivotRow := ReferenceRow;
      for Row := ReferenceRow + 1 to Dimen do
        if ABS(Coefficients[Row, ReferenceRow]) >
          ABS(Coefficients[PivotRow, ReferenceRow]) then
          PivotRow := Row;
      if PivotRow <> ReferenceRow then
      begin
        DummyRow := Coefficients[PivotRow];
        Coefficients[PivotRow] := Coefficients[ReferenceRow];
        Coefficients[ReferenceRow] := DummyRow;
        Dummy := Constants[PivotRow];
        Constants[PivotRow] := Constants[ReferenceRow];
        Constants[ReferenceRow] := Dummy;
      end
      else { If the diagonal element is zero, no solution exists  }
        if ABS(Coefficients[ReferenceRow, ReferenceRow]) <
          NearlyZero then Error := 2;
      if Error = 0 then
        for Row := ReferenceRow + 1 to Dimen do
          { Make the ReferenceRow element of these rows zero }
          if ABS(Coefficients[Row, ReferenceRow]) > NearlyZero then
          begin
```

```
        Multiplier := -Coefficients[Row, ReferenceRow] /
          Coefficients[ReferenceRow, ReferenceRow];
        EROmultAdd(Multiplier, Dimen, Coefficients[ReferenceRow],
          Coefficients[Row]);
        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;

end.
```

```
unit EQ;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Grids, Graph, ComCtrls, Buttons;
const ArraySize = 60; { Maximum number of equations }
type complex = record
    Re, Im: double;
  end;
  CompVector = array[0..ArraySize] of complex;
  IntVector = array[0..ArraySize] of integer;
  RealVector = array[1..ArraySize] of double;
  matrix = array[1..ArraySize] of RealVector;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
    Button1: TButton;
    Button6: TButton;
    Button3: TButton;
    Panel1: TPanel;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    TabSheet3: TTabSheet;
    Timer1: TTimer;
    Label1: TLabel;
    Label2: TLabel;
    Panel2: TPanel;
    Graph: TXYGraph;
    Label3: TLabel;
    Panel3: TPanel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    ErrorGraph: TXYGraph;
    Button4: TButton;
    Button5: TButton;
    Button7: TButton;
    Panel4: TPanel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Button2: TButton;
    Button8: TButton;
    BitBtn1: TBitBtn;
    procedure FormCreate(Sender: TObject);
```

```
    procedure Main();
    procedure Timer1Timer(Sender: TObject);
    function EquF(w: double): double;
    function DifF(x: double): double;
    procedure MinMax(LeftEndPoint: double;
      RightEndPoint: double;
      var Answer: double;
      var Error: byte);
    procedure Partial_Pivoting(Dimen: integer;
      Coefficients: matrix;
      Constants: RealVector;
      var Solution: RealVector;
      var Error: byte);
    procedure ShowGroupDelay(h: integer);
    procedure ShowError(h: integer);
    procedure ShowFirst();
    procedure ShowEnd();
    procedure Initialize();
    procedure FindError();
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
var Coefficients: matrix;
  Constants, Solution, a, b, w: RealVector;
  e, D0, D, Tol, min, max, mm: double;
  Dimen, Row, Column, i, iter: integer;
  Error: byte;
  IterM, second: boolean;

function TForm1.EquF(w: double): double;
var D: double;
  i: integer;
begin
{ Function of Delay Equalization }
  D := 0;
  for i := 1 to (Dimen shr 1) do
```

```
    D := D + ((2 * a[i]) / (sqr(a[i]) + sqr(w - b[i])) +
      (2 * a[i]) / (sqr(a[i]) + sqr(w + b[i])));
  EquF := D / (2 * pi);
end;

function TForm1.DifF(x: double): double;
const Tolerance = 1E-10;
type vector = array[1..100] of double;
var Iter, TwoToTheIterMinus2, Extrap: integer;
  DeltaX, FourToTheExtrapMinus1: double;
  OldEstimate, NewEstimate: vector;

  function EvaluateFirstDeriv(X: double;
    DeltaX: double): double;
  var LeftPoint, RightPoint: double;
  begin
    LeftPoint := EquF(X - DeltaX);
    RightPoint := EquF(X + DeltaX);
    EvaluateFirstDeriv := (RightPoint - LeftPoint) / (2 * DeltaX);
  end;

begin
  if ABS(X) < Tolerance then
    DeltaX := Sqrt(Tolerance)
  else
    DeltaX := ABS(X * Sqrt(Tolerance));
  OldEstimate[1] := EvaluateFirstDeriv(X, DeltaX);
  Iter := 1;
  TwoToTheIterMinus2 := 1;
  repeat
    Iter := Succ(Iter);
    DeltaX := DeltaX / 2;
    NewEstimate[1] := EvaluateFirstDeriv(X, DeltaX);
    TwoToTheIterMinus2 := TwoToTheIterMinus2 * 2;
    FourToTheExtrapMinus1 := 1;
    for Extrap := 2 to Iter do
    begin
      FourToTheExtrapMinus1 := FourToTheExtrapMinus1 * 4;
      NewEstimate[Extrap] :=
        (FourToTheExtrapMinus1 * NewEstimate[Extrap - 1]
        - OldEstimate[Extrap - 1]) / (FourToTheExtrapMinus1 - 1);
    end;
    OldEstimate := NewEstimate;
  until (ABS(NewEstimate[Iter - 1] - NewEstimate[Iter]) <=
    ABS(Tolerance * NewEstimate[Iter])) or
    (ABS(DeltaX) < Tolerance);
  DifF := NewEstimate[Iter];
end;
```

```
procedure TForm1.MinMax(LeftEndPoint: double;
  RightEndPoint: double;
  var Answer: double;
  var Error: byte);
const NearlyZero = 1E-10;
var Tol, yLeft, yRight, MidPoint, yMidPoint: double;
  Iter, MaxIter: integer;
  Found: boolean;

  function TestForRoot(X, OldX, Y, Tol: double): boolean;
  begin
    TestForRoot := (ABS(Y) <= NearlyZero) or (ABS(X - OldX) < ABS(OldX * Tol));
  end;

begin
  Maxiter := 500;
  Tol := 1E-10;
  Error := 0;
  Found := false;
  yLeft := DifF(LeftEndpoint);
  yRight := DifF(RightEndpoint);
  if ABS(yLeft) <= NearlyZero then
  begin
    Answer := LeftEndpoint;
    Found := true;
  end;
  if ABS(yRight) <= NearlyZero then
  begin
    Answer := RightEndpoint;
    Found := true;
  end;
  if not Found then
  begin
    if yLeft * yRight > 0 then
      Error := 2;
    if Tol <= 0 then
      Error := 3;
    if MaxIter < 0 then
      Error := 4;
  end;
  if (Error = 0) and (Found = false) then
  begin
    Iter := 0;
    yLeft := DifF(LeftEndPoint);
    while not (Found) and (Iter < MaxIter) do
    begin
      Iter := Succ(Iter);
      MidPoint := (LeftEndpoint + RightEndpoint) / 2;
      yMidPoint := DifF(MidPoint);
      Found := TestForRoot(MidPoint, LeftEndpoint, yMidPoint, Tol);
```

```
      if (yLeft * yMidPoint) < 0 then
        RightEndpoint := MidPoint
      else
      begin
        LeftEndpoint := MidPoint;
        yLeft := yMidPoint;
      end;
      Answer := MidPoint;
    end;
    if Iter >= MaxIter then
      Error := 1;
  end;
end;

procedure TForm1.Partial_Pivoting(Dimen: integer;
  Coefficients: matrix;
  Constants: RealVector;
  var Solution: RealVector;
  var Error: byte);
const NearlyZero = 1E-10;
var Sum, Multiplier, Dummy: double;
  Term, Row, ReferenceRow, PivotRow: integer;
  DummyRow: RealVector;

  procedure EROmultAdd(Multiplier: double;
    Dimen: integer;
    var ReferenceRow: RealVector;
    var ChangingRow: RealVector);
  var Term: integer;
  begin
    for Term := 1 to Dimen do
      ChangingRow[Term] := ChangingRow[Term] +
        Multiplier * ReferenceRow[Term];
  end;

begin
  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
        { Make Coefficients matrix upper triangular }
    ReferenceRow := 0;
    while (Error = 0) and (ReferenceRow < Dimen - 1) do
```

```
    begin
      ReferenceRow := Succ(ReferenceRow);
        { Find row with largest element in this column }
        { and switch this row with the ReferenceRow   }
      PivotRow := ReferenceRow;
      for Row := ReferenceRow + 1 to Dimen do
        if ABS(Coefficients[Row, ReferenceRow]) >
          ABS(Coefficients[PivotRow, ReferenceRow]) then
          PivotRow := Row;
      if PivotRow <> ReferenceRow then
      begin
        DummyRow := Coefficients[PivotRow];
        Coefficients[PivotRow] := Coefficients[ReferenceRow];
        Coefficients[ReferenceRow] := DummyRow;
        Dummy := Constants[PivotRow];
        Constants[PivotRow] := Constants[ReferenceRow];
        Constants[ReferenceRow] := Dummy;
      end
      else { If the diagonal element is zero, no solution exists  }
        if ABS(Coefficients[ReferenceRow, ReferenceRow]) <
          NearlyZero then Error := 2;
      if Error = 0 then
        for Row := ReferenceRow + 1 to Dimen do
          { Make the ReferenceRow element of these rows zero }
          if ABS(Coefficients[Row, ReferenceRow]) > NearlyZero then
          begin
            Multiplier := -Coefficients[Row, ReferenceRow] /
              Coefficients[ReferenceRow, ReferenceRow];
            EROmultAdd(Multiplier, Dimen, Coefficients[ReferenceRow],
              Coefficients[Row]);
            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 TForm1.Initialize();
var h: integer;
begin
  Tol := 1E-10;
  iter := -1;
  IterM := false;
  FillChar(Coefficients, SizeOf(Coefficients), 0);
  FillChar(Constants, SizeOf(Constants), 0);
  Dimen := 10;
  e := 0.005;
  D0 := 0.55;
  a[1] := 1.3575; b[1] := 0.66667;
  a[2] := 1.3575; b[2] := 2;
  a[3] := 1.3575; b[3] := 3.3333;
  a[4] := 1.3575; b[4] := 4.6667;
  a[5] := 1.086; b[5] := 6;
  StringGrid1.Cells[0, 0] := '                p';
  StringGrid1.Cells[1, 0] := '                q';
  StringGrid2.Cells[0, 0] := '                  w[i]';
  StringGrid2.Cells[1, 0] := '                Delta D';
  Timer1.Interval := 0;
  main();
  FindError();
  h := Graph.MakeCurve('Original', clRed, 1, psDot, true);
  ShowGroupDelay(h);
  h := ErrorGraph.MakeCurve('Original', clRed, 1, psDot, true);
  ShowError(h);
  h := ErrorGraph.MakeCurve('0', clSilver, 1, psSolid, true);
  ErrorGraph.AddPoint(h, 0, 0);
  ErrorGraph.AddPoint(h, 6, 0);
  ErrorGraph.Paint;
  Label3.Visible:=false;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Initialize();
end;

procedure TForm1.Main();
var h: integer;
begin
  iter := iter + 1;
  for row := 1 to (Dimen shr 1) do
  begin
    StringGrid1.Cells[0, row] := FloatToStr(a[row]);
    StringGrid1.Cells[1, row] := FloatToStr(b[row]);
  end;
```

```
    for Row := 1 to Dimen do
    begin
     if odd(Row) then
     begin
      if Row = 1 then
      begin
       if not IterM then
        w[Row] := 0
       else
       begin
        w[Row] := b[1] / 3;
        if DifF(w[Row]) >= 0 then
         MinMax(0, w[Row], mm, Error)
        else
         MinMax(w[Row], b[1], mm, Error);
        if Error = 0 then w[Row] := mm;
       end;
      end
      else
      begin
       w[Row] := (b[(Row shr 1)] + b[(Row shr 1) + 1]) / 2;
       if IterM then
       begin
        if DifF(w[Row]) >= 0 then
         MinMax(b[(Row shr 1)], w[Row], mm, Error)
        else
         MinMax(w[Row], b[(Row shr 1) + 1], mm, Error);
        if Error = 0 then w[Row] := mm;
       end;
      end;
     end
     else
     begin
      w[Row] := b[Row shr 1];
      if IterM then
      begin
       if DifF(w[Row]) <= 0 then
        MinMax((b[(Row shr 1) - 1] + b[(Row shr 1)]) / 2, w[Row], mm, Error)
       else
        MinMax(w[Row], (b[(Row shr 1)] + b[(Row shr 1) + 1]) / 2, mm, Error);
       if Error = 0 then w[Row] := mm;
      end;
     end;
     StringGrid2.Cells[0, row] := FloatToStr(w[row]);
    end;
end;

procedure TForm1.FindError();
var h: integer;
begin
```

```
for Row := 1 to Dimen do
begin
  D := EquF(w[Row]);
  if odd(Row) then
  begin
    if (Row = 1) and not IterM then
    begin
      Constants[Row] := D0 - D;
      min := abs(D - D0);
    end
    else
    begin
      Constants[Row] := D0 - D - e / 2;
      min := abs(D - D0 + e / 2);
    end;
  end
  else
  begin
    Constants[Row] := D0 - D + e / 2;
    max := abs(D - D0 - e / 2);
  end;
  StringGrid2.Cells[1, row] := FloatToStr(Constants[row]);
end;
Label1.Caption := Format('%0.3d', [Iter]);
if (min < Tol) and (max < Tol) then
begin
  if IterM then
    showEnd()
  else
  begin
    showfirst();
    Button6.Visible := true;
    Button1.Enabled := false;
    Button2.Enabled := false;
  end;
end;
for Row := 1 to Dimen do
begin
  for Column := 1 to Dimen do
  begin
    if Column <= (Dimen shr 1) then
    begin
      i := Column;
      Coefficients[Row, Column] :=
        2 * ((sqr(w[Row] - b[i]) - sqr(a[i]))
        / sqr(sqr(a[i]) + sqr(w[Row] - b[i]))
        + (sqr(w[Row] + b[i]) - sqr(a[i]))
        / sqr(sqr(a[i]) + sqr(w[Row] + b[i])));
    end
    else
```

```
      begin
        i := Column - (Dimen shr 1);
        Coefficients[Row, Column] :=
          4 * a[i] * ((w[Row] - b[i]) /
          sqr(sqr(a[i]) + sqr(w[Row] - b[i])) -
          (w[Row] + b[i]) / sqr(sqr(a[i]) + sqr(w[Row] + b[i])));
      end;
    end;
  end;
  Partial_Pivoting(Dimen, Coefficients, Constants, Solution, Error);
  if Error = 0 then
  begin
    for i := 1 to (Dimen shr 1) do
    begin
      a[i] := a[i] + Solution[i];
      b[i] := b[i] + Solution[(Dimen shr 1) + i];
    end;
  end
  else
    MessageDlg('***** Error *****', mtInformation,
      [mbOk], 0);
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Main();
  FindError();
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Main();
  FindError();
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Timer1.Interval := 5;
end;

procedure TForm1.ShowFirst();
var h: integer;
begin
  Label3.Caption := '*** First Iteration ***';
  Label3.Visible := true;
  Label5.Caption := '.....' + IntToStr(Iter) + 'th Iteration';
  Label5.Visible := true;
  Label8.Caption := '.....' + IntToStr(Iter) + 'th Iteration';
  Label8.Visible := true;
  IterM := true;
```

```
  Timer1.Interval := 0;
  h := Graph.MakeCurve('First Iteration', clLime, 1, psDot, true);
  ShowGroupDelay(h);
  second := True;
end;

procedure TForm1.ShowEnd();
var h: integer;
begin
  Label3.Caption := '**End of Iteration**';
  Label3.Visible := true;
  Label6.Caption := '.....' + IntToStr(Iter) + 'th Iteration';
  Label6.Visible := true;
  Label9.Caption := '.....' + IntToStr(Iter) + 'th Iteration';
  Label9.Visible := true;
  Timer1.Interval := 0;
  h := Graph.MakeCurve('Second Iteration', clFuchsia, 1, psDot, true);
  ShowGroupDelay(h);
  h := ErrorGraph.MakeCurve('Second Iteration', clFuchsia, 1, psDot, true);
  ShowError(h);
  Button1.Enabled:= false;
  Button2.Enabled:= false;
end;

procedure TForm1.ShowGroupDelay(h: integer);
var
  x: real;
  gd: double;
begin
  begin
    x := 0;
    while x <= 6 do
    begin
      GD := EquF(x);
      Graph.AddPoint(h, x, gd);
      x := x + 0.001;
    end;
    Graph.Paint;
  end;
end;

procedure TForm1.ShowError(h: integer);
var
  x: real;
  gd: double;
begin
  begin
    for row := 1 to 10 do
    begin
      ErrorGraph.AddPoint(h, w[row], constants[row] * 1E3);
```

```
    end;
    ErrorGraph.Paint;
  end;
end;

procedure TForm1.Button6Click(Sender: TObject);
var h: integer;
  x: real;
  gd: double;
begin
  Main();
  FindError();
  h := ErrorGraph.MakeCurve('Group delay', clLime, 1, psDot, true);
  for row := 1 to 10 do
  begin
    ErrorGraph.AddPoint(h, w[row], constants[row] * 1E3);
  end;
  ErrorGraph.Paint;
  Button1.Enabled := true;
  Button2.Enabled := true;
  Label3.Caption := '.....READY....';
  Button6.visible := false;

end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Button5.Click();
  Button8.Click();
  Button1.Enabled:=true;
  Button2.Enabled:=true;
  Initialize();
end;

end.
```