โครงการวิจัยทุนวิจัยรัชดาภิเษกสมโภช

รายงานฉบับสมบูรณ์
"การพัฒนาแบบจำลองการสั่นสะเทือน
เนื่องจากการระเบิด"

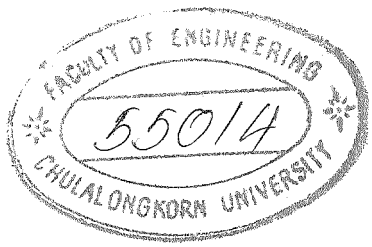
FINAL REPORT
"DEVELOPMENT OF MODELS FOR VIBRATION
FROM BLASTING"

โดย

ดร. สง่า ตั้งชวาล และคณะ

มิถุนายน 2538

การพัฒนาแบบจำลองการสั่นสะเทือน เนื่องจากการระเบิด



โครงการวิจัยทุนวิจัยรัชดาภิเษกสมโภช
ประจำปีงบประมาณ 2537

รายงานฉบับสมบูรณ์ปีแรกเรื่อง

การพัฒนาแบบจำลองการสั่นสะเทือน
เนื่องจากการระเบิด

คณาจารย์ที่ปรึกษาโครงการ
ศาสตราจารย์ ดร. ปณิธาน ลักคุณะประสิทธิ์
รองศาสตราจารย์ ดร. ขวัญชัย ลีเผ่าพันธ์
ผู้ช่วยศาสตราจารย์ ดร. ทวี ธนะเจริญกิจ

คณะผู้วิจัย
รองศาสตราจารย์ ดร. สง่า ตั้งชวาล
รองศาสตราจารย์ ฉลับ ปัทมสูต
รองศาสตราจารย์ มัณฑนา ปราการสมุทร

เสนอต่อ คณะกรรมการติดตามและประเมินผลโครงการวิจัยทุนรัชดาภิเษกสมโภช
จุฬาลงกรณ์มหาวิทยาลัย

เสนอรายงาน มิถุนายน 2538

บทคัดย่อ

การพัฒนาแบบจำลองการสั่นสะเทือน เนื่องจากการระเบิด ปีแรก

การสั่นสะเทือนเนื่องจากคลื่นพัลส์ที่เป็นคลื่นกระแทกเนื่องจากการระเบิดได้ถูกนำมาพิจารณาเพื่อเป็นแนวทางพัฒนาแบบจำลองการสั่นสะเทือนชั่วครู่ ระบบที่ถูกเลือกมีทั้งระบบ Single Degree of Freedom กับระบบ Multidegree of Freedom มีการวิเคราะห์เชิง numerical เพื่อหา solution ที่เหมาะสมที่สุดของคลื่นกระแทกที่แสดงเป็น response spectrum โปรแกรมแบบจำลองที่เขียนเพื่อจำลองทั้งของระบบ SDF และ N-DOF เป็น package program โดยใช้ภาษา C^{++} ซึ่งมีการทำงานภายใต้ Microsoft Windows version 3.1 Thai Edition

ABSTRACT

Development of Models for Vibration from Blasting : Phase I

Vibration from detonated wave pulse which is shock wave was reviewed in order to develop models for transient vibration. Selected systems were Single Degree of Freedom System and Multidegree of Freedom System. Numerical analyses were derived to find the optimum solution of shock wave that represents as response spectrum. Modeling programs were written for both SDF and N-DOF systems. It is in the form of package program by using C^{++} language and runs under Microsoft Windows version 3.1 Thai Edition.

คำนำ

รายงานฉบับสมบูรณ์ปีแรกของโครงการวิจัยเงินทุนรัชดาภิเษกสมโภชของจุฬาลงกรณ์มหาวิทยาลัย เรื่อง "การพัฒนาแบบจำลองการสั่นสะเทือนเนื่องจากการระเบิด" ได้จัดทำผลงานปีแรกเสร็จสิ้นตามทฤษฎีและผลการวิเคราะห์ที่ได้บรรยายไว้ในรายงาน

การพัฒนาแบบจำลองเป็นไปใน 2 แนวทาง กรณีของแนวทางที่หนึ่งเป็นการจำลองระบบ Single Degree of Freedom ของคลื่นพัลส์ที่เกิดชั่วครู่ กับอีกอีกแนวทางหนึ่งเป็นการจำลองระบบ Multidegree of Freedom ของคลื่นพัลส์ที่เกิดชั่วครู่ คณะผู้วิจัยได้รวบรวมแนวทาง วิเคราะห์เชิงคณิตศาสตร์ขั้นสูงไว้ก่อนจะนำ solutions ที่มีผู้ค้นคิดไว้นำมาใช้เป็นแนวทางในการพัฒนาโปรแกรมแบบจำลอง ซึ่งเป็นส่วนของงานวิจัยปีแรก ในการวิจัยปีต่อไปจะได้นำผลการวัดการสั่นสะเทือนในภาคสนามที่ได้จากการระเบิดหินเพื่อใช้ในงานวิศวกรรมเป็นข้อมูลดิบที่ป้อนเข้าไปเพื่อทดสอบ แก้ไข ปรับปรุงแบบจำลองให้เหมาะสมสำหรับงานระเบิดหินในประเทศไทย

คณะผู้วิจัย
โครงการ "การพัฒนาแบบจำลองการสั่นสะเทือน
เนื่องจากการระเบิด"

มิถุนายน 2538

กิตติกรรมประกาศ

คณะผู้วิจัยขอขอบคุณต่อคณะกรรมการวิจัยเงินทุนวิจัยรัชดาภิเษกสมโภชของจุฬาลงกรณ์มหาวิทยาลัยที่ให้ความสนับสนุนทางด้านค่าใช้จ่ายมา ณ ที่นี้ด้วย

งานวิจัยโครงการนี้มีผู้ช่วยวิจัยหลายท่าน ซึ่งคณะผู้วิจัยขอขอบคุณเป็นชุดแรก ได้แก่ คณาจารย์ที่ปรึกษาโครงการ อาจารย์ไพบูลย์ ศรีภคากร อาจารย์ประจำภาควิชาเครื่องกลที่ช่วยตรวจสอบความถูกต้องของ derived solutions ขอขอบคุณสำหรับ คุณนิสิต อินต๊ะชัย แห่งบริษัท Four Seas Supply กับนาย ดำรงค์ แซ่ก้วย นิสิตปริญญาโทภาควิชาวิศวกรรมคอมพิวเตอร์ที่เขียน packaged program ชุดแรกสำหรับระบบ SDF

ในส่วนของงานวิจัยชุดที่สอง ขอขอบคุณต่อนาย ปริญญา วิลัยธรรม กับนาย ชาตรี ศรีชัย บัณฑิตปริญญาตรีวิศวกรรมเหมืองแร่สำหรับการรวบรวมข้อมูลภาคสนาม สำหรับการเขียน packaged program ชุดสองสำหรับระบบ N-DOF คณะผู้วิจัยขอขอบคุณต่อนายพันธ์ศักดิ์ ทีฆสกุล บัณฑิตปริญญาตรีวิศวกรรมคอมพิวเตอร์/เหมืองแร่

กรณีของงานพิมพ์คณะผู้วิจัยขอขอบคุณต่อคุณวรวรรณ วิงประวัติ คุณทัศนี แก้วเจริญ คุณสัมฤทธิ์ บุญจิตร แห่งบริษัท Sinobrit ที่เหนื็ดเหนื่อยในการพิมพ์และแก้ไขต้นฉบับหลายครั้ง

สารบัญ

รายการตารางประกอบ

รายการรูปประกอบ

หน้า

หน้า

หน้า

หน้า

หน้า

สัญกรณ์ (NOTATION) และสัญลักษณ์ (SYMBOL)

A	=	displacement amplitude
$\overline{A}$	=	dynamic matrix
$\tilde{A}$	=	square symmetric matrix
$[\overline{A}S]$	=	deflated matrix
a	=	acceleration of the motion
a_{max}	=	maximum acceleration
$[a]$	=	flexibility matrix
α	=	logarithmic decrement = $\ln\frac{x_1}{x_2}$, also constant in matrix orthogonalization
β	=	critical damping fraction = c_1/c_c
c	=	longitudinal (dilatation) wave velocity
c_1	=	damping coefficient of the system (dashpot)
c_c	=	critical damping
c_i, c_j, c_k	=	values of damping coefficient for each system
Δ	=	spring deformation (static)
δ	=	differential movement = x - u and also as a unit impulse or a delta function
δ_{max}	=	maximum differential movement
$\delta(t)_{max}$	=	peak of the relative displacement for impulse excitation
$\dot{\delta}_{max}$	=	maximum differential velocity
$\ddot{\delta}$	=	acceleration of the mass relative to the ground
η	=	time interval for impulse excitation
F_0	=	excitation force for step function
F(t)	=	applied force (impulse), and also a solution of the equation
$\hat{F}$	=	impulse excitation force
f	=	frequency of the motion
f(t)	=	arbitrary force
h	=	response time interval (increment) = Δt

$h(t)$	=	response function to a unit impulse
I	=	inverse of the upper triangular matrix
K	=	stiffness matrix
k_{ii}	=	generalized stiffness
k	=	spring stiffness
L	=	wave motion in radial direction
Λ	=	diagonal matrix of eigenvalues = $[\lambda_i]$
λ	=	eigenvalue
$\overline{\lambda}$	=	eigenvalue (iteration)
M	=	mass matrix
M_{ii}	=	generalized mass
m	=	mass of the system
N	=	numbers of degree of freedom
ω	=	circular frequency = $2\pi f$
ω_d	=	damped natural circular frequency
ω_n	=	undamped natural circular frequency
P	=	primary or longitudinal wave, also modal matrix
P^T	=	transpose of modal matrix
$\widetilde{P}$	=	modal matrix (orthonormal mode)
$\widetilde{P}^T$	=	transpose of orthonormal mode for modal matrix
$\varnothing$	=	eigenvector
$\varnothing_i(x)$	=	normal mode
$\varnothing_i^T$	=	transpose mode
$\widetilde{\varnothing}_i$	=	orthonormal mode
R	=	Rayleigh wave, also rotation matrix
r	=	radial distance from explosion
S	=	shear or transverse (distorsion) wave, also another expression for sweeping matrix
$[S]$	=	sweeping matrix
T	=	wave motion in transverse direction
t	=	real time

τ	=	a variable (time) for integration of the wave pulse
τ_d	=	damped natural period
τ_n	=	undamped natural period
θ	=	rotation angle of transformation coordinate
U	=	transformation matrix
U^{-1}	=	standard form of transformation coordinate
U^T	=	transpose of transformation matrix
u	=	ground particle displacement
u_0	=	initial displacement for time history
$\dot{u}$	=	ground particle velocity
$\dot{u}_{max}$	=	maximum particle velocity
$\ddot{u}$	=	ground particle acceleration
V	=	wave motion in vertical direction
X	=	amplitude of oscillation for the viscous damping, also a mode vector
X_{max}	=	maximum amplitude of the damped system
x	=	absolute displacement of the mass, and also as the "x" axis in cartesian coordinate
x_n	=	amplitude of "n" free oscillation
x_{n+1}	=	amplitude of "n+1" free oscillation
$\dot{x}$	=	absolute velocity of the mass
$\ddot{x}$	=	absolute acceleration of the mass
Y	=	transformed displacement vector
y	=	one of the axis for cartesian coordinate
z	=	one of the axis for cartesian coordinate

บทที่ 1
บทนำ

คลื่นพัลส์ที่เดินทางในช่วงระยะเวลาสั้นๆ ในระหว่างที่มีการจุดระเบิด (detonation) ในหินหรือวัสดุอื่นที่ใกล้เคียง จะเป็น impact pressure ทำให้เกิดการสั่นสะเทือน (vibration) ในหินหรือตัวกลาง อันที่ผลกระทบต่อสภาวะสิ่งแวดล้อมทางด้านประสาทเสียงและความรู้สึกเป็นอย่างมาก นอกจากนี้ยังมีผลทำให้เกิดการร้าวหรือความล้า (fatigue) ในโครงสร้างของสิ่งก่อสร้างได้

ลักษณะทั่วไปของคลื่นจากการระเบิดในงานวิศวกรรมแบ่งออกเป็น 3 ประเภทใหญ่ๆ คือ compressive, shear และ surface wave จากประเภทของคลื่นทั้งสาม ยังสามารถจำแนกย่อยออกเป็น 2 varieties คือ body wave ซึ่งมีการเดินทางผ่านเข้าไปในวัสดุที่เป็นตัวกลาง (หินหรือดิน) และ surface wave ซึ่งมีการเดินทางผ่านแนวพื้นผิว (ปกติเป็นพื้นผิวส่วนบน) คลื่น surface wave ที่สำคัญที่สุด คือ Rayleigh (R-wave)

ผลกระทบของการระเบิดในระยะทางใกล้ๆ คลื่นที่มีผลกระทบมากคือ body wave ซึ่งจะมีการเดินทางในวัสดุอยู่ในรูปแบบของ spherical manner จนกระทั่งมันกระทบ interface ซึ่งเป็นขอบเขตต่อเนื่องระหว่างชั้นหิน ดิน หรือบนพื้นผิวดิน ที่จุดตัดนี้ shear และ surface wave จะเกิดขึ้นมา และ Rayleigh surface wave นี้จะมีผลกระทบมากขึ้นที่ระยะทางไกลขึ้น

การแปลความหมายชนิดของคลื่นจะยุ่งยากมากในช่วงระยะใกล้ๆ กับจุดระเบิด ในขณะที่ในระยะไกลๆ คลื่นที่เดินทางช้า เช่น shear และ surface wave จะแยกออกมาจาก compressive wave ทำให้จำแนกชนิดของคลื่นได้ง่ายขึ้น คลื่นทั้ง 3 แบบนี้จะให้แบบแผนของการเคลื่อนที่ในอนุภาคของดินและหินแตกต่างกันไป ดังนั้นโครงสร้างของสิ่งก่อสร้าง (เช่น อาคาร ตึก) จะมีการเปลี่ยนรูปและปริมาตร (deformation) แตกต่างกันไปขึ้นอยู่กับชนิดของคลื่นที่ผ่าน

จากการศึกษาทางด้าน seismology พบว่าการเดินทางของ surface wave แบบ cylindrical propagation ใน infinite, elastic และ homogeneous material จะมีค่า seismic velocity ประมาณ 0.9 เท่าของ body wave และ ค่ายอดสูงสุดของแอมพลิจูดของ surface wave acceleration, velocity และ displacement จะมีการสลาย (decay) เป็นสัดส่วนเท่ากับค่า $(1/r)^{0.5}$ เมื่อกำหนดให้ r เป็นระยะทางที่คลื่นเดินทางผ่าน (Selberg, 1952) ค่ายอดสูงสุดของแอมพลิจูดแบบ spherical propagation จะมีการสลายเป็นสัดส่วนเท่ากับค่า $(1/r)^{n}$ โดยที่ค่า n มีสองค่า ในระยะทางใกล้ๆซึ่ง

เป็นกรณีของ static equilibrium ค่า n มีค่าเท่ากับ 2 และ ที่ระยะไกลขึ้น ซึ่งเป็นกรณีของ dynamic equilibrium ค่า n จะลดลงไปเรื่อยๆ จนเท่ากับ 1

การที่จะพยายามหาวิธีที่สามารถจะนำมาประกอบในการพัฒนาแบบจำลอง (model) ของ transient wave parameters เพื่อที่จะสามารถหาแนวโน้มที่ก่อให้เกิดการเสียหาย (damage potential) ของการสั่นสะเทือน (vibration) ในหินหรือตัวกลางอันทีผลกระทบต่อสภาวะสิ่งแวดล้อมทางด้านประสาท เสียง และความรู้สึกเป็นอย่างมาก นอกจากนี้ยังมีผลกระทบทำให้เกิดการแตกร้าว (crack) หรือความล้า (fatigue) ในโครงสร้างของอาคารข้างเคียงหรือสิ่งก่อสร้างที่ใกล้เคียงกับบริเวณที่ทำการระเบิดได้

บทที่ 2

การสั่นสะเทือนแบบ SINGLE DEGREE OF FREEDOM

การคาดคะเนระดับของการสั่นสะเทือน ไม่ว่าคลื่นประเภทไหนมักจะทำการตรวจวัด ตามแนวรัศมีของหลุมเจาะระเบิด ในเชิงปฏิบัติการตรวจวัดคลื่นการสั่นสะเทือน มักจะใช้เครื่องบันทึกความสั่นสะเทือนแบบอัตโนมัติ ซึ่งเรียกรวมๆกันเป็นแบบ self-recording displacement seismo-graph แนวโน้มของผลกระทบของคลื่นการสั่นสะเทือนมีต่อโครงสร้างอาคารสิ่งก่อสร้าง เช่น ความเร็วของอนุภาค ความเร่งของอนุภาค ควรจะมีพิสัยในระดับที่เหมาะสม

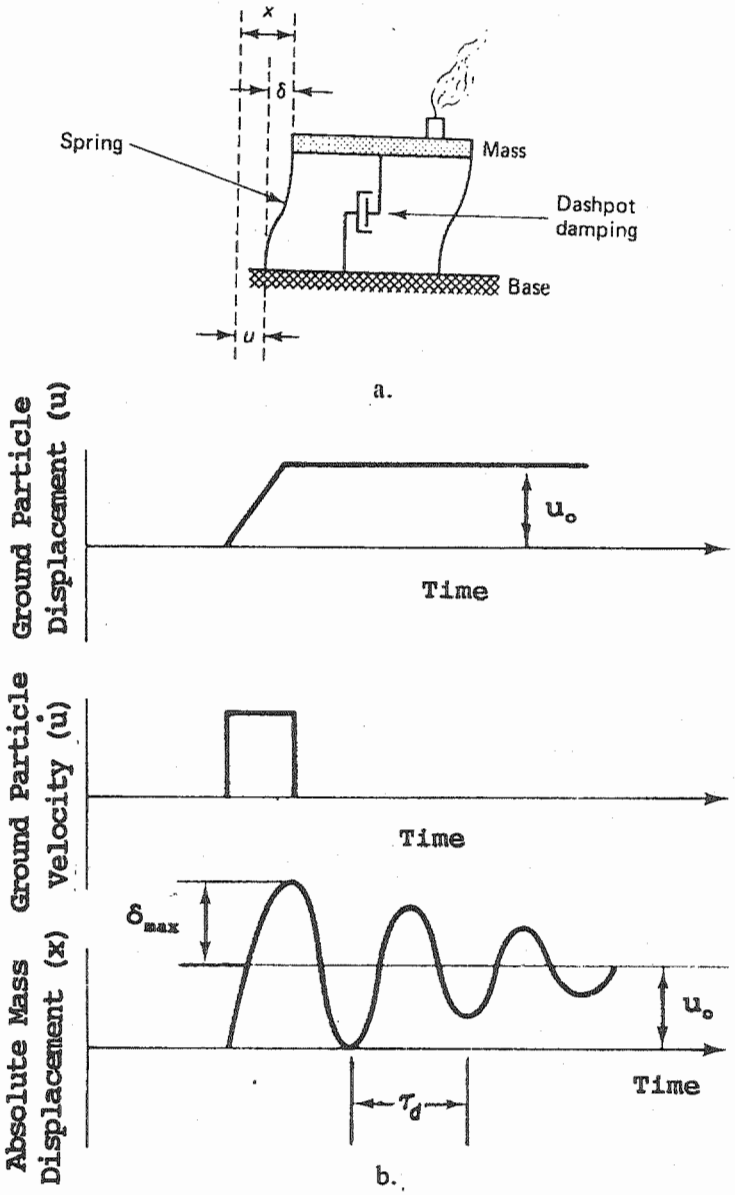
2.1 แนวทางวิเคราะห์แบบจำลอง SDF

ผลเสียหายของรอยแตกร้าวของสิ่งก่อสร้าง เป็นผลมาจากการตอบสนองของอาคารโครงสร้าง (response of structure) ที่มีต่อคลื่นการสั่นสะเทือนที่ผ่านมาในอาคารโครงสร้างนั้นๆ ลักษณะของโครงสร้างที่ถูกกระทำต่อโหลด (load) ของการสั่นสะเทือน ได้แก่

1. มวล (mass) ของส่วนประกอบที่สำคัญ (main components)
2. สติฟเนส (stiffness) ของส่วนประกอบที่สำคัญ
3. ปริมาณของพลังงาน (energy) ที่แผ่กระจายและก่อให้เกิด differential movement ในรอยแตกเล็กๆ (crack) รอยต่อ (joint) และส่วนอื่นๆ

แบบจำลองเพื่อหาอันตรกริยา (interaction) เชิงพลศาสตร์ใน 3 หัวข้อข้างต้นก็อาศัยระบบของ Single Degree of Freedom หรือใช้ตัวย่อว่า ระบบ SDF ดังแสดงในรูปที่ 1 หน้า 4 ในกรณีของรูปที่ 1a concentrate mass มีความคล้ายคลึงกับมวลของส่วนประกอบที่สำคัญ ส่วนสปริงแทนได้เท่ากับสติฟเนสของส่วนประกอบที่สำคัญ และ dashpot ที่มีคุณสมบัติของความต้านทานเชิงหนืด (viscous resistance) แทนแบบจำลองในการแผ่กระจายพลังงาน ส่วนรูปที่ 1b เป็นแบบจำลองเชิงคณิตศาสตร์ของอาคารโครงสร้างสำหรับค่าตัวแปรเชิงอิลาสติกต่างๆ

ถ้าหากจำแบบจำลองนี้ไปเทียบกับอาคารโครงสร้าง concentrate mass จะคล้ายคลึงกับมวลของ floor structure สำหรับสปริงในแนวดิ่งคล้ายคลึงกับ wall structure และตัว dashpot ดูดกลืนพลังงานได้เช่นเดียวกับโครงสร้างอาคารจริงจากการเปรียบเทียบพฤติกรรมของอาคารที่อยู่อาศัยที่มีความสูงไม่เกิน 3 ชั้น พบว่ามีความคล้ายคลึงกับระบบของ Single Degree of Freedom เมื่อมีการ



รูปที่ 1 แสดงแบบจำลองของ Single Degree of Freedom (SDF)

a. แบบจำลองเชิงคณิตศาสตร์ของอาคารโครงสร้าง

δ = ค่าของ differential movement

x = ค่าของ absolute displacement ของมวล

u = ค่าของ absolute displacement ของพื้นผิวดิน

b. แสดง time history ของการบันทึกค่าต่างๆ ได้แก่ ค่าการเปลี่ยนตำแหน่งสัมบูรณ์ของพื้นผิวดิน (u) ค่าความเร็วของอนุภาคพื้นผิวดิน ($\dot{u}$) ค่าการเปลี่ยนตำแหน่งสัมบูรณ์ของมวล (x)

δ_{max} คือค่าของ maximum differential movement

เคลื่อนไหว (movement) เพียงทิศทางเดียว ในทางตรงกันข้าม เมื่ออาคารที่อยู่อาศัยมีความสูงเกิน 3 ชั้น ก็จำเป็นต้องใช้แบบจำลองของ Multidegree of Freedom

2.2 สมการการเคลื่อนที่ของ SDF

สมการของการเคลื่อนที่สำหรับ Single Degree of Freedom เมื่อมีการกระตุ้นของพื้นผิวดิน (Dowding, 1985 ; Meirovitch, 1986, 1992) ได้แก่

$$m \cdot \frac{\partial^2 x}{\partial t^2} + c_1 \cdot \frac{\partial \delta}{\partial t} + k \cdot \delta = 0 \qquad (1)$$

โดยที่

$\frac{\partial^2 x}{\partial t^2}$ = ค่าความเร่งสัมบูรณ์ของมวล (มวลสารมีค่าเท่ากับ m)

c_1 = ค่าสัมประสิทธิ์ของ damping

$\frac{\partial \delta}{\partial t}$ = ค่าความเร็วของมวลที่สัมพัทธ์กับพื้นผิวดิน

k = ค่าคงที่ของ linear spring

δ = ค่าการเปลี่ยนตำแหน่งสัมพัทธ์ระหว่างพื้นผิวดินและมวล ($\delta = x - u$)

เมื่อแทนค่าของ differential movement, δ ลงในสมการที่ 1 จะได้เป็น

$$m \cdot \frac{\partial^2 \delta}{\partial t^2} + c_1 \cdot \frac{\partial \delta}{\partial t} + k \cdot \delta = -m \cdot \frac{\partial^2 u}{\partial t^2} \qquad (2)$$

จากสมการของการเคลื่อนที่ ค่า circular natural frequency ของ undamped spring-mass system, ω_n (Thomson, p. 18) ได้แก่

$$\omega_n = (k/m)^{1/2} \tag{3}$$

ในกรณีของ viscously damped free vibration จากภาพไดอะแกรมของ free-body ดังแสดงในรูปที่ 2 (คัดลอกมาจาก Thomson p. 29) ได้พิสูจน์ว่าค่าของ critical damping, c_c ในกรณีพิเศษที่เกิดขึ้นระหว่าง oscillatory motion และ nonoscillatory motion เมื่อ $(c_1/2m)^2 = k/m$, and radical is zero

$$c_c = 2m(k/m)^{1/2} = 2m\omega_n = 2(k.m)^{1/2} \tag{4}$$

เมื่อกำหนดค่า β เท่ากับ fraction of critical damping

$$\beta = \frac{c_1}{2(k \cdot m)^{1/2}} \tag{5}$$

ถ้าหากว่ามวลถูกทำให้เคลื่อนที่จากจุดสมดุล การแกว่ง (oscillate) จะไม่เกิดเมื่อปล่อยมวล แต่มันจะคืนกลับสู่สภาพที่จุดสมดุล เมื่อ $c_1 = 2(k.m)^{1/2}$ หรือเรียกว่าเกิด critically damped

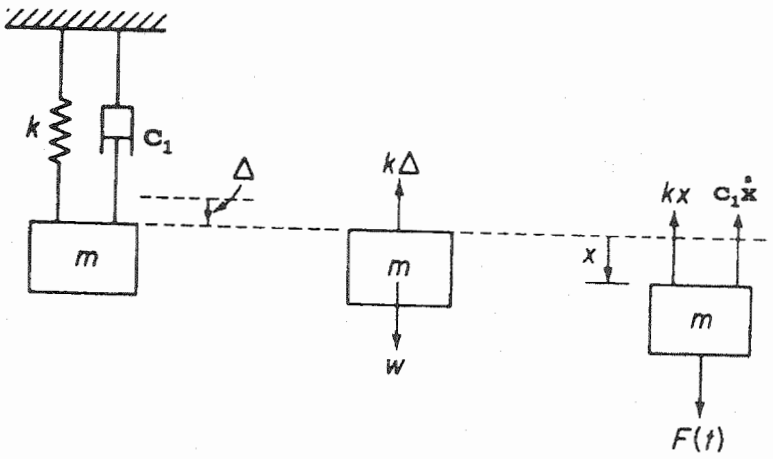
ดังนั้น ถ้าให้ ω_d = frequency ของ damped oscillation นั่นคือ

$$\omega_d = \omega_n(1 - \beta^2)^{1/2} \tag{6}$$

สมการที่ 2 จึงสามารถเขียนใหม่ได้เป็น

$$\frac{\partial^2 \delta}{\partial t^2} + 2\beta\omega_n \cdot \frac{\partial \delta}{\partial t} + \omega_n^2 \delta = -\frac{\partial^2 u}{\partial t^2} \tag{7}$$

เมื่อเทียบในเทอมเปอร์เซ็นต์ของ critical damping (β) กับเปอร์เซ็นต์ของ circular natural frequency (ω_n) จะได้ผลบันทึกของ ground-acceleration time history ที่ต้องการ integrate จากจุดที่เวลาเป็นศูนย์ (0) จนถึงเวลาเท่ากับ t ได้ค่าอยู่ในรูปของ $\frac{\partial^2 u(t)}{\partial t^2}$



รูปที่ 2 ภาพแสดง free-body diagram ของ viscously damped free vibration

c_1 = damping coefficient of the dashpot

k = spring stiffness

m = mass

x = single coordinate ของการเคลื่อนที่

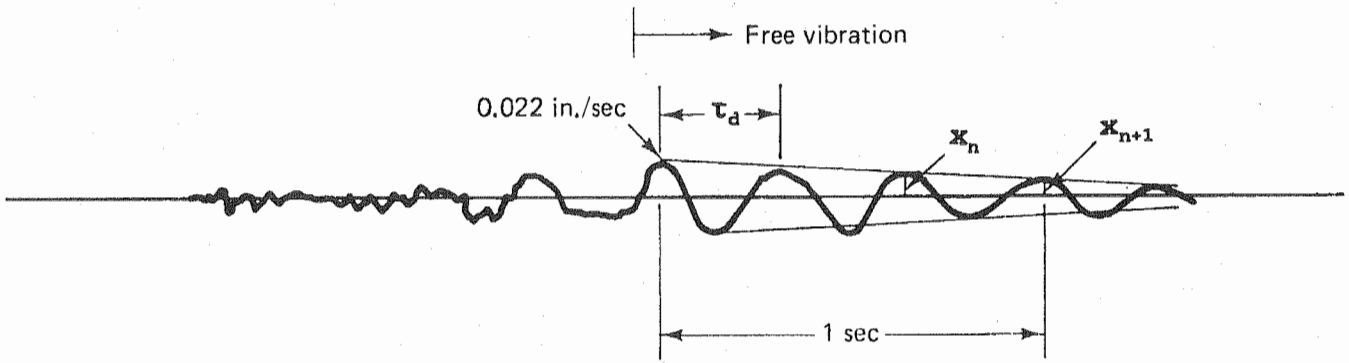
F(t) = Excitation force

Δ = spring deformation in the static equilibrium position

ในเชิงปฏิบัติเมื่อโครงสร้างมีค่า undamped natural frequency(นั่นคือ ถ้าหากทราบค่า ω_n และ β)ก็ไม่จำเป็นต้องทราบค่าของ m, k และ c_1 เพื่อจำลองแบบผลกระทบต่ออาคารโครงสร้างให้ถูกต้องแม่นยำ นอกจากนี้ในการหาการตอบสนองของอาคารโครงสร้าง เมื่อทำการคำนวณ free vibration time history คุณสมบัติเชิงพลศาสตร์ของ ω_n และ β สามารถหาได้ใกล้เคียงแม่นยำมากกว่าหาจากค่าของ m, k และ c_1 เหตุผลที่สำคัญอีกอย่างคือพารามิเตอร์เหล่านี้ (m, k, c_1) มีปัญหาที่ยากจะแจกแจง เพราะมีองค์ประกอบอื่นเข้ามากระทบการวัดค่าของ m, k, c_1 ตัวอย่างเช่น degree of fixity of the columns (มีผลต่อ k) และ damping coefficient, c_1

2.3 ตัวอย่างผลกระทบในเรื่องการสั่นสะเทือนเนื่องจากการระเบิดเชิง SDF

Dowding (1971) ได้ทำการวัด free vibration ที่มีต่อเป้าหมายที่เป็นอาคารโครงสร้าง ผลของการบันทึกดังแสดงไว้ในรูปที่ 3 หน้าที่ 8 เป็นการเฝ้าสังเกตการณ์ (monitoring) ของ horizontal



รูปที่ 3 ผลการบันทึกของ velocity time-history ที่มีต่ออาคารโครงสร้างอันเนื่องมาจาก free vibration (จากงานของ Dowding, 1971)

velocity time-history ที่บริเวณจุดชั้นที่ 6 ของอพาร์ตเมนท์สูง 6 ชั้น ในแนวทิศทางขนานต่อ short axis ของอาคารโครงสร้าง

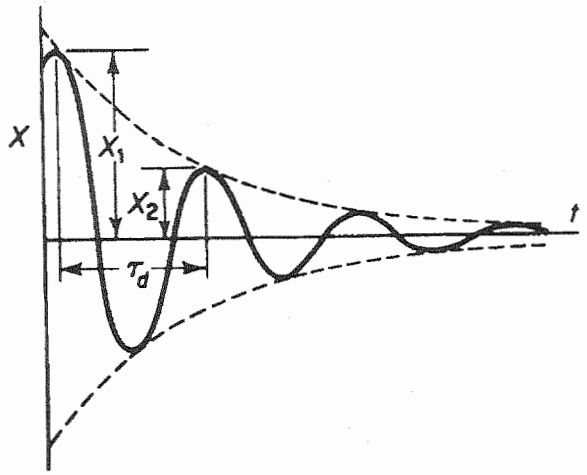
Damped natural period for first mode of vibration (τ_d) มีความสัมพันธ์กันเมื่อเทียบกับ undamped natural frequency (ω_n) ซึ่งได้แสดงไว้ในสมการที่ 3 และ 6 เมื่อนำมาเรียบเรียงใหม่ จะได้

$$\tau_d = \frac{2\pi}{\omega_d} = \frac{2\pi}{\omega_n(1 - \beta^2)^{1/2}} \qquad (8)$$

จะเห็นได้ว่า เมื่อค่า β เล็กมาก ω_n จะมีค่าใกล้เคียงหรือเท่ากับ ω_d ซึ่งสอดคล้องกับผลการวิจัยของ Newmark and Hall (1969) ที่พบว่า damping fractions สำหรับ undamaged structure มีค่า ประมาณ 2-3 % นั่นคือ ω_n จะมีค่าเท่ากับ ω_d สำหรับอาคารที่อยู่อาศัยที่มีความสูงต่ำกว่า 3 ชั้น

การที่จะหาปริมาณของ damping ที่อยู่ในระบบ วิธีการที่สะดวกก็จะวัดจากอัตราการสลายตัวของ free oscillation โดยทั่วไปพบว่ายิ่งมี damping สูงมาก อัตราการสลายตัวก็จะเป็นไปด้วยความรวดเร็ว ซึ่งสามารถแสดงได้ในรูปของ logarithmic decrement (ดูรูปที่ 4)

ค่าของ logarithmic decrement (α) ได้ถูกนิยามว่าเป็น natural logarithm ของอัตราส่วนของแอมพลิจูดที่ต่อเนื่องกัน 2 ชุด ใดๆ นั่นคือ



รูปที่ 4 อัตราการสลายตัวของ oscillation เมื่อแสดงในรูปของ logarithmic decrement

$$\alpha = \ln \frac{x_1}{x_2} \qquad (9)$$

Thomson (1993) ได้แสดงสรุปหาความสัมพันธ์ระหว่างค่าของ logarithmic decrement กับ damped period เป็น

$$\alpha = \beta \omega_n \tau_d \qquad (10)$$

เมื่อแทนค่า damped period, $\tau_d = 2\pi / \omega_n (1 - \beta^2)^{1/2}$ ก็จะได้ค่าของ logarithmic decrement เปลี่ยนเป็น

$$\alpha = \frac{2\pi\beta}{(1 - \beta^2)^{1/2}} \qquad (11)$$

สมการที่ 11 ข้างบนนี้เกือบเป็น exact solution ถ้าหาก β มีค่าน้อยค่า $(1 - \beta^2)^{1/2}$ ก็จะมีค่าประมาณใกล้เคียงหรือเกือบเท่ากับ 1 ซึ่งจะได้ approximate equation เป็น

$$\alpha \cong 2\pi\beta \qquad (12)$$

หรือเขียนใหม่ว่า critical damping fraction จะสามารถหาจากการสลายตัวของ free oscillation ของอาคารโครงสร้างโดยมีความสัมพันธ์เท่ากับ

$$\beta = \frac{1}{2\pi}\left(-\ln \frac{x_{n+1}}{x_n}\right) \qquad (13)$$

โดยที่ x_{n+1} และ x_n เป็น successive amplitudes

2.4 แนวทางหาผลกระทบโดยอาศัย SDF Response Spectra

จากที่กล่าวมาในหัวข้อ 2.3 การบันทึกการตอบสนอง (response) ของอาคารโครงสร้างต่อคลื่นการสั่นสะเทือน สรุปได้ว่าการเคลื่อนที่ของอนุภาคในพื้นผิวดิน (ground motion) มีผลต่อพฤติกรรมของระบบ และแสดงออกในลักษณะของ response spectra

สมมุติว่ามีแรงภายนอก F (รูปที่ 5 a) ซึ่งแปรผันกับเวลาและรู้ทิศทางของการเคลื่อนที่ พารามิเตอร์ หรือ quantity ที่จะระบุการเคลื่อนที่ของระบบ คือ การเปลี่ยนตำแหน่ง, x (ดูรูปที่ 5b) ในเมื่อต้องการจะหากราฟที่แสดงพฤติกรรมของระบบที่มีคุณสมบัติพื้นฐานที่จำเป็น ได้แก่ m, k, c_1 (m = mass ; k = spring stiffness ; c_1 = damping coefficient) เมื่อได้ผลของการบันทึก

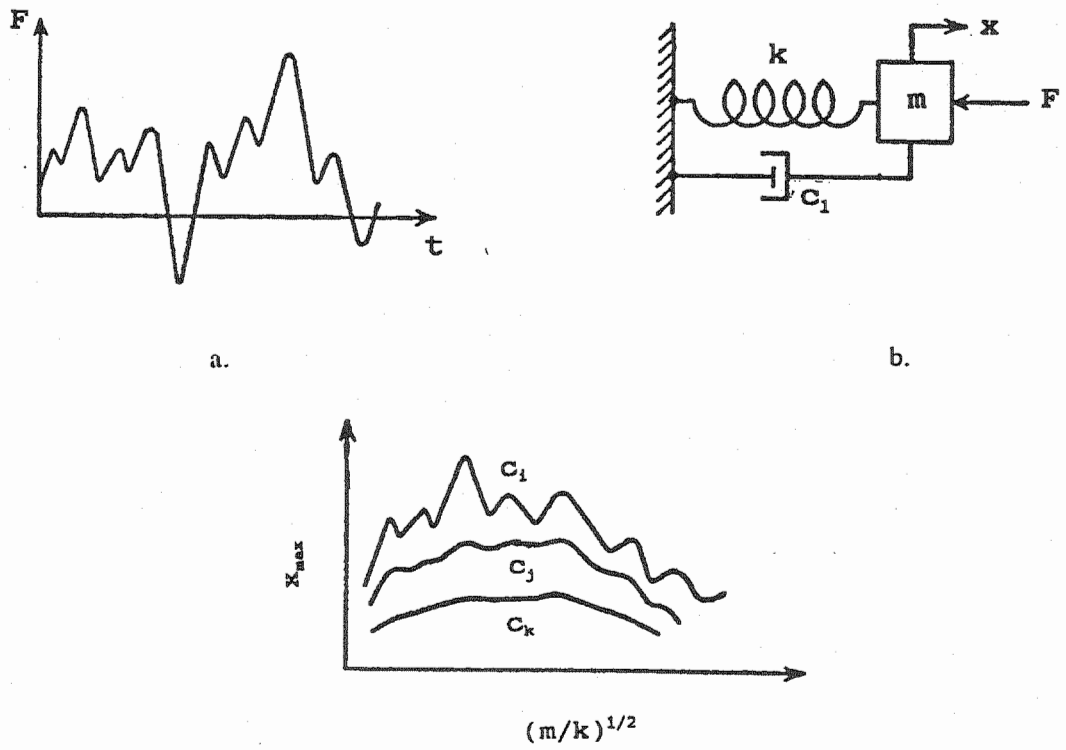
แล้วค่าสูงสุด ที่แสดงเป็นค่ายอดสูงสุด (peak) ของการเปลี่ยนตำแหน่งของอนุภาค, x_{max} พลอตเทียบกับพารามิเตอร์ หรือ quantity $(m/k)^{1/2}$ สำหรับค่า damping coefficient ที่มี หลายค่า (ในรูปที่ 5c แสดงเป็น c_i, c_j, c_k) จึงได้เซตของกราฟที่เป็น response spectum ของแรง F (ในกรณีของแต่ละค่าของ damping coefficient เรียกว่า response spectrum ถ้ามีมากกว่าหนึ่งค่าเรียกว่า response spectra) แต่เนื่องจากค่า $(m/k)^{1/2}$ เป็น natural period of vibration ของระบบ ในกรณีของระบบพื้นฐาน ค่า $c_i = 0$ หรือ period เป็นส่วนกลับ (reciprocal) ของ natural frequency ดังนั้น response sprectrum เกิดในลักษณะนี้จึงเรียกว่า frequency response curves

แนวคิดของ response spectra มาจากการเน้นปัญหาทางพลศาสตร์ เมื่อมีแรงกระทำภายนอก ในหลายกรณีวิศวกรอาจตัดสินใจได้ดีกว่า ถ้ารู้พฤติกรรมของอาคารโครงสร้างที่แสดงเป็น response spectra จะสามารถระบุถึงสถานภาพของพฤติกรรมของโครงสร้างแต่มีข้อจำกัดว่า ระบบต้องเป็น single degree of freedom และการตอบสนองทั้งหมด (total response) ของระบบที่ยากขึ้น สามารถประมาณได้จาก response spectra โดยอาศัย superposition อย่างไรก็ตาม วิธีการนี้ไม่ได้ exact solution ทั้งนี้เพราะว่า response spectrum แต่ละกราฟ ไม่ได้ให้ข้อมูลเชิง phase relationships ระหว่างการเคลื่อนที่ในหลาย modes of vibration การที่ผู้วิเคราะห์ข้อมูลตั้งสมมุติฐานว่า การตอบสนองสูงสุดในหลาย modes เกิดขึ้นในช่วงเวลาเดียวกันอาจจะเป็นการ overestimate ของการตอบสนองทั้งหมดของระบบ

อาคารโครงสร้างมีสิ่งประกอบหลายอย่าง และที่สำคัญที่สุด คือ ผนัง (wall) และ superstructure skeleton ตัวอย่างการตอบสนองต่อคลื่นจากการระเบิดได้จากงานของ Medearis (1978) และ Dowding and Corser (1981) ได้แสดงไว้ในรูปที่ 6 หน้า 13 บริเวณโครงสร้างของ superstructure ไว้วาง transducer 1ตัว ไว้ตรงมุมที่จุด i ในรูปที่ 6b และวาง transducer อีก 1 ตัว ตรงกึ่งกลางผนังตรงจุด ii ตัว transducer ทั้งคู่นี้จะวัดค่าการเคลื่อนที่สัมบูรณ์

การเคลื่อนที่ของ superstructure ตรงมุมที่ประกบกัน คือการเคลื่อนที่ที่เกี่ยวข้องกับ shearing และ torsional distorsion ของ frame (ดังแสดงในรูปที่ 6c) ในขณะที่การเคลื่อนที่ของผนังตรงจุดกึ่งกลางเกี่ยวกับของ bending ของผนังที่ถูกคลื่นมากระทำ

ในรูปที่ 6a ผนัง และ superstructure มีการสั่นสะเทือนแบบ free vibration ภายหลังที่ ground motion ผ่านพ้นไป การเคลื่อนที่ของผนังมีแอมพลิจูดใหญ่(สูง)กว่าการเคลื่อนที่ของ superstructure



รูปที่ 5 response spectra ต่อแรงที่กระทำภายนอก

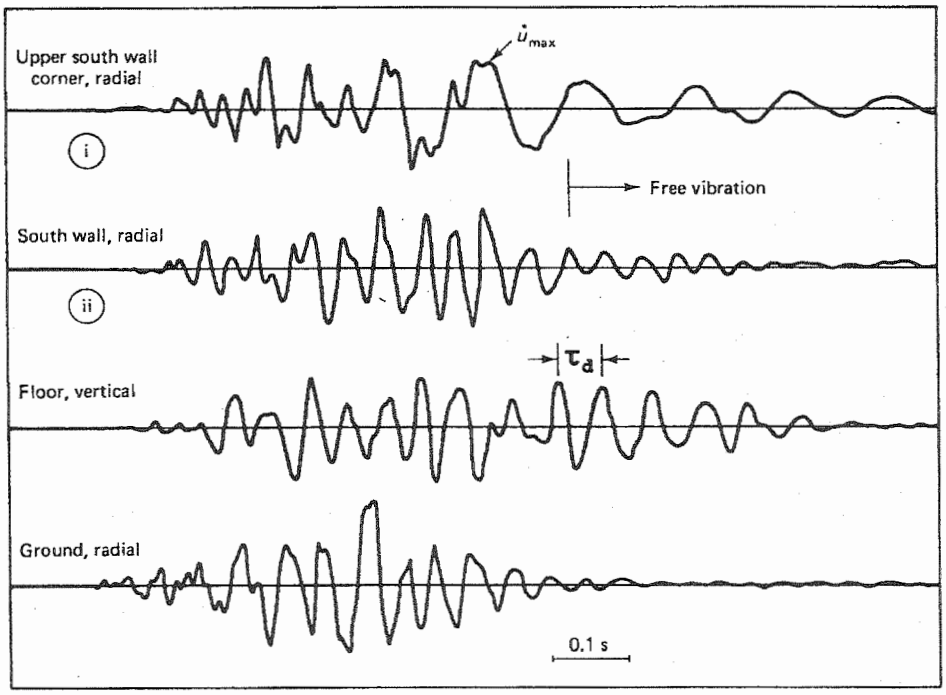
a. การบันทึกผลของ ground motion

b. แบบจำลองอย่างง่ายของ viscously damped free vibration

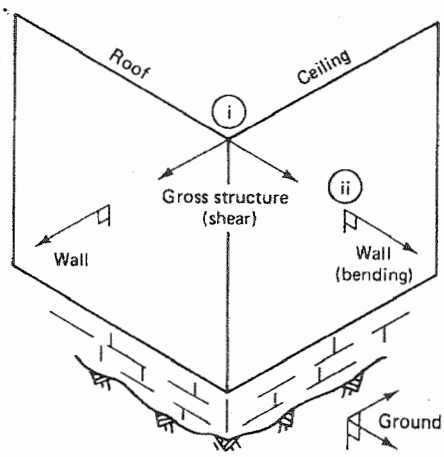
c. กราฟที่พลอตค่าของจุดยอดสูงสุดของการเปลี่ยนตำแหน่งกับค่าของสัมประสิทธิ์ของ damping system (คัดลอกจาก Hudson and Housner, 1957)

สิ่งที่น่าสังเกตเพิ่มเติมก็คือการสั่นสะเทือนของผนังมีความถี่สูงกว่าของ superstructure ในขณะเกิด free vibration ค่าประมาณของความถี่ประมาณของความถี่ของผนัง จะมีพิสัยตั้งแต่ 12 ถึง 20 เฮิร์ทซ์ สำหรับพิสัยความถี่ของ superstructure มีตั้งแต่ 5 ถึง 10 เฮิร์ทซ์

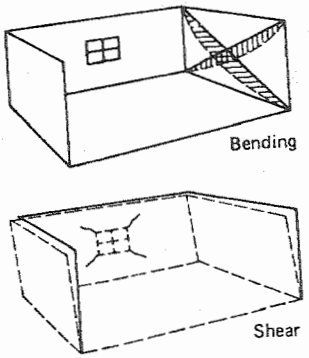
จุดยอดสูงสุดของการเคลื่อนที่จะลดลงอย่างรวดเร็วในขณะที่มี free response ยิ่งมีการสลายตัวในอัตราสูงในช่วงสั้นๆ ก็แสดงว่า โครงสร้างของสิ่งก่อสร้างมีความหน่วง (damping) สูง มักพบว่าพิสัยของ critical damping สำหรับโครงสร้างมีค่าตั้งแต่ 2 เปอร์เซ็นต์ ถึง 10 เปอร์เซ็นต์



a.



b.



c.

รูปที่ 6 การตอบสนองของสิ่งก่อสร้างต่อคลื่นจากการระเบิด

a. การเคลื่อนที่คลื่นการสั่นสะเทือนที่ผ่านเข้ามาใน wall และ superstructure

b. อุปกรณ์ transducer ที่ทำวางไว้ที่ตำแหน่ง i และ ii เพื่อหาทิศทางของการไหวสะเทือน

c. การเกิด bending ของ wall และการ shear ของ superstructure

บทที่ 3
การสั่นสะเทือนชั่วครู่เนื่องจากอิมพัลส์

ในเชิงพลศาสตร์เมื่อระบบถูกกระตุ้นอย่างทันทีทันใด จะเกิดการตอบสนองต่อแรงที่มากระทำ ซึ่งเป็นลักษณะของการตอบสนองชั่วครู่ (transient response) การแกว่ง (oscillation) ที่เกิดขึ้นที่ความถี่ธรรมชาติ จะมีแอมพลิจูดแปรผันพึ่งพิงกับชนิดของสิ่งที่กระตุ้นภายนอก

3.1 ธรรมชาติของการกระตุ้นเนื่องจากอิมพัลส์

อิมพัลส์ (impulse) เป็น time integral ของแรงภายนอกที่มากระทำ ได้ความสัมพันธ์เมื่อวิเคราะห์เรื่องการกระตุ้น คือ

$$\hat{F} = \int F(t)\, dt \tag{14}$$

โดยที่ $\hat{F}$ = impulsive force

F(t) = Excitation

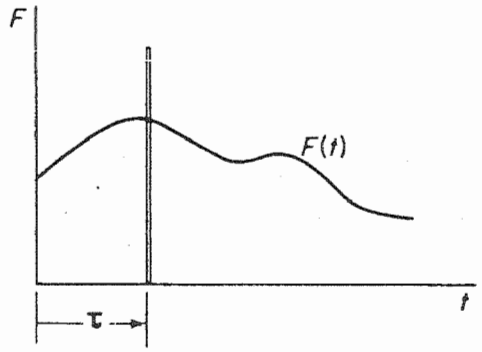
ค่า impulsive force มักมีขนาดใหญ่ (สูง) มาก และกระทำในช่วงระยะเวลาสั้นๆ โดยมี time integral เป็น finite จึงเรียกแรงประเภทนี้ว่า แรงอิมพัลส์ (impulsive force) แรงอิมพัลส์ที่มีขนาด $\hat{F}/\eta$ ที่มี time duration เท่ากับ η ในขณะที่ η เข้าใกล้ศูนย์ แรงก็จะอยู่ในภาวะ infinite

อย่างไรก็ตาม นิยามของอิมพัลส์, $\hat{F}$ มีค่าของ time integral เป็น finite เมื่อ $\hat{F}$ มีค่าเป็น unity แรงที่มีข้อจำกัดนี้จะเรียกว่า หน่วยอิมพัลส์ (unit impulse) หรือในทางคณิตศาสตร์แทนได้เท่ากับ delta function (ในกรณีที่ค่า η เข้าใกล้ศูนย์) ค่า delta function ในขณะที่ $t = \tau$ จะแสดงเป็น expression ในรูปสัญลักษณ์ของ $\delta(t - \tau)$ และมีคุณสมบัติดังสมการที่ 15 คือ

$$\left.\begin{aligned} \delta(t-\tau) &= 0 \text{ สำหรับทุกค่าที่ } t \neq \tau \\ &= \text{มีค่าสูงมาก เมื่อ } t = \tau \\ \int_0^\infty \delta(t-\tau)dt &= 1.0 \text{ เมื่อกำหนดให้ } 0 < \tau < \infty \end{aligned}\right\} \tag{15}$$

ถ้า $\delta(t - \tau)$ ถูกคูณโดย time function f(t) ดังแสดงในรูปที่ 7 หน้า 15 ดังนั้นผลคูณจะมีค่าเท่ากับศูนย์ ยกเว้นเมื่อ $t = \tau$ และค่า time integral เป็น

$$\int_0^{\infty} f(t)\delta(t-\tau)\,dt = f(\tau) \qquad 0 < \tau < \infty \tag{16}$$



รูปที่ 7 ฟอร์มการเคลื่อนที่ของการสั่นสะเทือนชั่วครู่ (transient vibration) ที่มีต่ออิมพัลส์ที่มากระทำ

จากกฎของนิวตัน Fdt = mdv เมื่อมีอิมพัลส์, $\hat{F}$ กระทำต่อมวล, m มีผลก่อให้เกิดการเปลี่ยนแปลงความเร็วอย่างทันที่ทันใด (มีค่าเท่ากับ $\hat{F}$ /m) โดยไม่มีการเปลี่ยนตำแหน่ง (displacement) ที่ชัดเจน ภายใต้สภาวะ free vibration พบว่าระบบ undamped spring-mass ที่มีสภาวะเริ่มต้นที่ x(0) และ $\dot{x}(0)$ มีพฤติกรรมดังนี้

$$x = \frac{\dot{x}(0)}{\omega_n}\sin\omega_n t + x(0)\cos\omega_n t \tag{17}$$

ดังนั้น การตอบสนองของระบบ spring-mass ที่อยู่นิ่งขณะเริ่มต้น และถูกกระตุ้นด้วยอิมพัลส์ $\hat{F}$ คือ

$$x = \frac{\hat{F}}{m\omega_n}\sin\omega_n t = \hat{F}h(t) \tag{18}$$

โดยที่ h(t) เป็นค่าการตอบสนอง (response) ต่อหน่วยอิมพัลส์

$$h(t) = \frac{1}{m\omega_n}\sin\omega_n t \tag{19}$$

Thomson (pp. 28-31, 93) สรุปการหา solution ของ free vibration ที่เกิด damping ที่กำหนดสภาวะเริ่มต้น (initial condition) เมื่อ x(0) = 0 ได้ solution เป็น

$$x = \frac{\dot{x}(0)\cdot e^{-\beta\omega_n t}}{\omega_n(1-\beta^2)^{1/2}}\cdot\sin(1-\beta^2)^{1/2}\omega_n t \qquad (20)$$

แทนค่าสำหรับ สภาวะเริ่มต้นที่ $\dot{x}(0) = \hat{F}/m$ จะได้ค่าสมการดังนี้

$$x = \frac{\hat{F}}{m\omega_n(1-\beta^2)^{1/2}}\cdot e^{-\beta\omega_n t}\cdot\sin(1-\beta^2)^{1/2}\omega_n t \qquad (21)$$

จะเห็นได้ว่าค่า h(t) ซึ่งเป็นผลการตอบสนองของหน่วยอิมพัลส์มีความสำคัญมากในแก้ปัญหาคลื่นที่เกิดในภาวะชั่วครู่ (transient wave) ไม่ว่าจะเป็น กรณีของ damped หรือ undamped สมการการตอบสนองของอิมพัลส์จะอยู่ในรูปแบบของ

$$x = \hat{F}h(t) \qquad (22)$$

ซึ่งสมการข้างบนนี้เป็นส่วนพจน์ที่อยู่ทางด้านขวาของสมการ 18 หรือ สมการ 21

3.2 การกระตุ้นเชิงพลศาสตร์พื้นฐาน

ปกติการกระตุ้นเชิงพลศาสตร์ (dynamical excitation) จะมีปรากฏการณ์ที่มีการเคลื่อนที่อย่างทันทีทันใด ซึ่งเป็นผลมาจากการเปลี่ยนตำแหน่งของอนุภาค ความเร็วของอนุภาค และความเร่งของอนุภาค และสมการของการเคลื่อนที่สามารถจะแสดงออกในรูปแบบของการเปลี่ยนตำแหน่งสัมพัทธ์ (relative displacement, δ) โดยที่มีค่าของ $\delta = x - u$

สมการการเคลื่อนที่ (equation of motion) เชิงพลศาสตร์พื้นฐานสำหรับค่าจะเหมือนกับที่ได้แสดงไว้ในสมการ 7 หน้า 6 (ในเรื่องของ SDF) ซึ่งนำมาเขียนในแบบฟอร์มใหม่คือ

$$\ddot{\delta} + 2\beta\omega_n\dot{\delta} + \omega_n^2\delta = -\ddot{u} \qquad (23)$$

ผลจากระบบถูกกระตุ้นจากแรงงานภายนอก (external force-excited system) สามารถนำมาใช้กับการเคลื่อนที่ของระบบการกระตุ้นพื้นฐาน (base-excited system) เมื่อค่า F/m ถูกแทนที่ด้วย $-\ddot{u}$ หรือเป็นค่าลบของความเร่งพื้นฐาน

3.2.1 Response spectra ของ damped system

ในเรื่อง Single Degree of Freedom ได้สมการที่ 7 หัวข้อ 2.2 เป็นสมการเคลื่อนที่ในระบบ damped system โดยที่ Veletsos and Newmark (1964) ได้หา solution ของ relative displacement [$\delta(t)$] ที่เวลาใดๆ (t) จากนั้นหาความสัมพันธ์ในรูปแบบของ Duhamel's integral ของ absolute ground-acceleration time history, [$\frac{\partial^2 u}{\partial t^2}$] หรือ $\ddot{u}$ นั่นคือ

$$\delta(t) = -\frac{1}{\omega_n(1-\beta^2)^{1/2}} \cdot \int_0^t \ddot{u}(\tau) \cdot e^{-\beta\omega_n(t-\tau)} \cdot \sin[\omega_d(t-\tau)]\, d\tau \qquad (24)$$

โดยที่ δ และ [$\frac{\partial \delta}{\partial t}$] มีค่าศูนย์ที่ t(0) หรือ t_0

$$\omega = (k/m)^{1/2}$$

$$\beta = c_1/2(k.m)^{1/2}$$

$$c_1 = 2(k.m)^{1/2}$$

สภาวะของผลกระทบในสมการที่ 24 นี้ มวลถูกเปลี่ยนตำแหน่ง (displace) จากตำแหน่งสมดุล มวลจะไม่มีการแกว่ง (oscillate) เมื่อถูกปล่อยแต่จะเป็นเพียงคืนกลับตำแหน่งที่อยู่ในสภาวะสมดุล ในสถานการณ์นี้เรียกว่าระบบถูกทำให้เกิดการหน่วงวิกฤต (critical damping) ได้ค่า ω_d เท่ากับ $\omega_n(1 - \beta^2)$ การคำนวณหาค่าของ ground-acceleration time-history จำเป็นต้อง integrate จากเวลา 0 ถึงเวลา t โดยใช้สัญลักษณ์ $\ddot{u}(\tau)$ โดยค่า τ เป็นตัวแปรของ integration

สมการที่ 24 ทำให้ได้ผลกระทบของ single degree of freedom ในเทอมของ reletive displacement (หรือใช้ศัพท์ทางวิศวกรรมโยธาว่าเป็น differential movement) ระหว่างพื้นผิวดิน (ground) และมวล (mass) ซึ่งศึกษาได้จาก ground-acceleration time-history

ถ้าหากผู้วิจัยมีความประสงค์จะใช้ velocity time-history เป็นข้อมูลเข้า (input data) ของ time-history ความสัมพันธ์ระหว่าง ground particle velocity or relative velocity ($\dot{u}$) และ relative displacement (δ) สามารถจะหาได้จากวิธีการ integrating by parts ของสมการที่ 24 แล้วนำเทอม

หลายเทอมมารวมกัน ก็จะได้ expression ของสมการใหม่เป็น

$$\hat{u}(t) = -\int_0^t \ddot{u}(\tau)\cdot e^{-\beta\omega_n(t-\tau)} \cdot \left\{ \cos\left[\omega_d(t-\tau)\right] + \frac{\beta}{(1-\beta^2)^{1/2}} \cdot \sin\left\{\omega_d(t-\tau)\right\} \right\} d\tau \qquad (25)$$

โดยที่ δ และ $\dot{\delta}$ มีค่าศูนย์ที่ t(0) หรือ t_0

3.2.2 Response spectra ของ undamped system

เมื่อระบบในเชิงพลศาสตร์ถูกกระทำให้เกิดการเปลี่ยนตำแหน่ง และเกิดความเร็วกับความเร่งอย่างทันทีทันใด ถ้าหากระบบนี้เป็นแบบ undamped system ถูกทำให้เคลื่อนที่จากเดิมที่ at rest Thomson (p. 96) ได้ให้ solution ของ relative displacement เป็น

$$\delta(t) = -\frac{1}{\omega_n}\int_0^t \ddot{u}(\tau)\cdot \sin\,\omega_n(t-\tau)\,d\tau \qquad (26)$$

3.3 สเปกตรัมการตอบสนองต่อคลื่นกระแทก (SRS)

ในหัวข้อ 3.2 ได้แสดงวิธีการแก้ปัญหาการตอบสนองต่อ undamped spring-mass system ที่มีต่อคลื่นพัลส์ที่มากระตุ้นในช่วงของ time duration เท่ากับ t_1 เมื่อกำหนดให้ t_1 มีค่าน้อยเมื่อเทียบกับค่าคาบธรรมชาติ (natural period, τ_n) ของ spring-mass oscillator ซึ่งการที่ช่วงเวลามีค่าน้อยมากนี้ก็เป็นปรากฏการณ์มาจากคลื่นกระแทก (shock wave) นั่นเอง

ระบบ SDF ของ spring-mass ที่ใช้ undamped oscillator ได้ถูกยกมาใช้เป็นตัวอย่าง โดยวิธีการที่จะหาการตอบสนองของคลื่นกระแทกสามารถหาจากสเปกตรัมซึ่งได้จากการพลอตระหว่างค่ายอดสูงสุดของการตอบสนอง (peak response) ที่เป็นค่าฟังก์ชันกับค่าคาบธรรมชาติของออสซิลเลเตอร์ เรียกชื่อสเปกตรัมชนิดนี้ว่า สเปกตรัมการตอบสนองของคลื่นกระแทก (shock response spectrum : SRS) จุดสูงสุดของ peak มักเรียกว่า "maximax" เป็นจุดเดี่ยวบนกราฟของ time response

ในเรื่องการกระตุ้นอย่างไม่เจาะจง [arbitrary excitation f(t)] กำหนดค่า impulse response [h(t)] สำหรับ undamped spring-mass oscillator

$$h(t) = \frac{1}{m\omega_n} \sin \omega_n t \tag{27}$$

ดังนั้นได้ค่ายอดสูงสุดของการตอบสนอง ซึ่งใช้ในการพลอต response spectrum เป็นสมการ

$$x(t)_{\max} = \left| \frac{1}{m\omega_n} \int_0^t f(\tau) \sin \omega_n (t - \tau)\, d\tau \right|_{\max} \tag{28}$$

ในกรณีที่มีคลื่นกระแทกก่อให้เกิดการเคลื่อนที่อย่างทันทีทันใด จะได้ค่า f(t) ถูกแทนที่ด้วย - ü(t) เนื่องจาก support point มีความเร่งเกิดขึ้นได้ความสัมพันธ์เหมือนกับสมการ 26 หน้า 18 หรือเขียนใหม่เป็นจุดยอดสูงสุดของ peak (ที่เรียก maximax) ได้เป็น

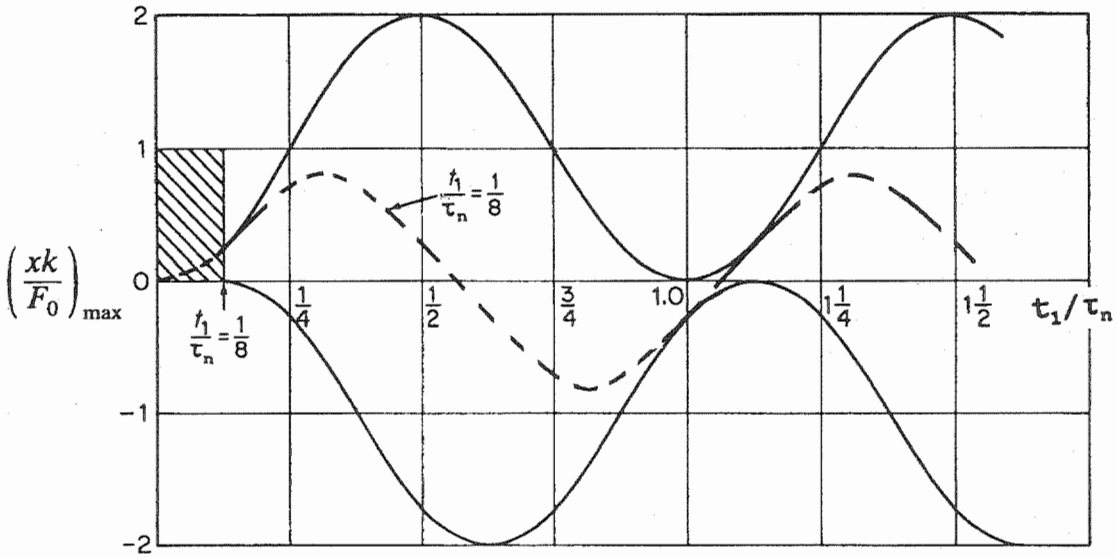
$$\delta(\tau)_{\max} = \left| \frac{-1}{\omega_n} \int_0^t \ddot{u}(\tau) \sin \omega_n (t - \tau)\, d\tau \right|_{\max} \tag{29}$$

กำหนดให้ τ_n เป็นคาบธรรมชาติของออสซิลเลเตอร์ค่าสูงสุดของ x(t) หรือ δ(t) จะหาได้จากการพลอตเมื่อเทียบกับของ t_1/τ_n ซึ่งจะมีค่า τ_n เป็นค่าคาบธรรมชาติของออสซิลเลเตอร์และ t_1 เป็นช่วงเวลาของคลื่นพัลส์ (ดูรูปที่ 8 หน้า 20)

จากการพลอตกราฟของ SRS ถ้ากำหนดให้คลื่นกระแทกเป็น คลื่นพัลส์สี่เหลี่ยมผืนผ้า (rectangular pulse wave) หาความสัมพันธ์ของ pulse excitation กับ rise time (Thomson, p. 102) ได้เป็น

$$\left(\frac{xk}{F_0}\right) = \{[1 - \cos \omega_n t] - [1 - \cos \omega_n (t - t_1)]\} \quad t > t_1 \tag{30}$$

จากสมการ 30 ข้างบนสำหรับคลื่นพัลส์รูปสี่เหลี่ยมผืนผ้า จะหาความสัมพันธ์ของ SRS ในรูปที่ 8 ได้ว่า มี step functions สองค่าที่เริ่มต้นที่ t = 0 และ $t = t_1$ ซึ่งค่าเหล่านี้ถูกพลอตในรูปที่ 8 สำหรับ $t_1/\tau_n = 1/8$ ความแตกต่างซึ่งเป็นการตอบสนองของออสซิลเลเตอร์สำหรับ $t > t_1$ ได้แสดงเป็นช่องลายและค่า peak response เท่ากับ $(xk/F_0) = 0.80$ ที่เวลา $t_m \cong 0.32\tau_n$ ดังนั้นจึงได้จุดแรกที่พลอตบนกราฟ SRS มีค่าเท่ากับ 0.80



รูปที่ 8 สเปกตรัมของการตอบสนองต่อคลื่นกระแทก (เส้นประ) ในกรณีที่มีค่า $t_1/\tau_n = 1/8$ ที่จุด $t_m \cong 0.32\tau_n$ และ $(xk/F_0)_{max} \cong 0.80$

ในทำนองเดียวกันก็สามารถได้กราฟ SRS ของคลื่นพัลส์รูปกึ่ง (ครึ่ง) ของกราฟไซน์ (half-sine wave pulse) กับคลื่นพัลส์รูปสามเหลี่ยม (triangular wave pulse) คล้ายกับการพลอตของคลื่นพัลส์รูปสี่เหลี่ยมผืนผ้า

บทที่ 4
การวิเคราะห์คลื่นเชิง NUMERICAL ของระบบ SDF

ในบางกรณีสมการเชิง differential ไม่สามารถจะทำการ integrate ได้ในสภาพ closed form จำเป็นต้องใช้การแก้ปัญหาของระบบด้วย numerical method สำหรับระบบยุ่งยากที่เป็น nonlinear หรือระบบที่ถูกกระตุ้นด้วยแรงภายนอกไม่สามารถแสดงออกเป็น simple analytic functions

4.1 วิธี Finite Difference

ในการวิเคราะห์เชิง numerical เพื่อแก้ปัญหาของ differential equation ที่ยุ่งยาก วิธีคำนวณเชิง finite difference นี้ใช้แทน continuous variable, t ด้วย discrete variable, t_i และ solve differential equation ได้ในช่วงของ time increment, h กำหนดให้ $h = \Delta t$ โดยเริ่มต้นจากสภาวะเริ่มต้นที่รู้ค่า

การแก้ปัญหาด้วยวิธี finite difference นี้ solution ที่ได้จะเป็นเพียงค่าประมาณ แต่ก็มีความแม่นยำที่สามารถยอมรับได้ในระดับที่มีค่าการเพิ่มของ time increment ช่วงเล็กๆ

ปัญหาของระบบพลศาสตร์ (dynamical system) ไม่ว่าเป็น linear หรือเป็น nonlinear จะสามารถใช้แทนได้ด้วย differential equation of motion ที่มีรูปฟอร์มทั่วไปเป็น

$$\left.\begin{aligned} \ddot{x} &= f(x, \dot{x}, t) \\ x_0 &= x(0) \\ \dot{x}_0 &= \dot{x}(0) \end{aligned}\right\} \qquad (31)$$

สมการที่ 31 ข้างบนนี้ค่าสภาวะเริ่มต้น x_0 และ x_0 จะต้องทราบค่า และต้องเป็นการเริ่มต้นที่ $t = 0$

4.1.1 การใช้ Finite Difference สำหรับ Undamped System

สมการเชิงพลศาสตร์ของระบบ undamped มีสมการทั่วไปเป็น

$$\left.\begin{aligned} \ddot{x} &= f(x, t) \\ x_0 &= x(0) \\ \dot{x}_0 &= \dot{x}(0) \end{aligned}\right\} \qquad (32)$$

กระบวนการต่อมามีชื่อเฉพาะที่เรียกว่า finite difference method ซึ่งอาศัยการพัฒนาเป็นรูปแบบของ Taylor expansion ของ x_{i+1} และ x_{i-1} รอบจุด pivotal point, i จะได้

$$\left.\begin{aligned} x_{i+1} &= x_i + h\dot{x}_i + \frac{h^2}{2}\ddot{x}_i + \frac{h^3}{6}\dddot{x}_i + \cdots \\ x_{i-1} &= x_i - h\dot{x}_i + \frac{h^2}{2}\ddot{x}_i - \frac{h^3}{6}\dddot{x}_i + \cdots \end{aligned}\right\} \tag{33}$$

เนื่องจากมี time interval คือ h = Δt เมื่อนำพจน์เหล่านี้มาลบกัน และเว้นค่าพจน์ที่เป็น higher order ทำให้ได้

$$\dot{x}_i = \frac{1}{2h}(x_{i+1} - x_{i-1}) \tag{34}$$

เมื่อนำพจน์มารวมกัน และเว้นค่าพจน์ที่เป็น higher order จะได้

$$\ddot{x}_i = \frac{1}{h^2}(x_{i-1} - 2x_i + x_{i+1}) \tag{35}$$

สำหรับสมการที่ 34 และ 35 ถ้าเว้นพจน์ของ h^2 จากนั้นแทนค่าด้วยสมการเชิง differential สมการที่ 32 และ 35 ที่ถูกจัดพจน์ใหม่เป็น

$$x_{i+1} = 2x_i - x_{i-1} + h^2 f(x_i, t_i) \qquad i \geq 2 \tag{36}$$

สมการ 36 ข้างบนนี้มีชื่อเรียกเฉพาะว่า "recurrence formula" สำหรับระบบ undamped นี้ เมื่อจะเริ่มต้นทำการคำนวณ ถ้าหากให้ i = 1 ในสมการ recurrence นี้จะสังเกตได้ว่าไม่ใช่เป็นการแก้ปัญหาแบบ self-starting นั่นคือ x_0 ทราบค่า แต่จำเป็นต้องใช้ค่า x_1 เพื่อหาค่า x_2 ดังนั้นเมื่อเริ่มต้นทำการคำนวณจำเป็นต้องหาสมการอื่นอีกสำหรับหา x_1 ซึ่งก็ใช้ Taylor's series ของสมการชุดแรก (ดูสมการที่ 33) หลังจากนั้นเว้นพจน์ที่เป็น higher order จะได้สมการสำหรับหา x_1 เป็น

$$x_1 = x_0 + h\dot{x}_0 + \frac{h^2}{2}\ddot{x}_0 = x_0 + h\dot{x}_0 + \frac{h^2}{2}f(x_0, t) \tag{37}$$

เมื่อทำไปเรื่อยๆจากสมการ 37 ช่วยให้หาค่า x_1 พจน์ของสภาวะเริ่มต้นซึ่งหลังจากนั้นค่า x_2, x_3, ... ก็สามารถหาได้จากสมการ 36 หน้า 22

ข้อควรระวังระหว่างการคำนวณเมื่อละเว้นพจน์ที่เป็น higher order จะมีผลทำให้เกิดปรากฏการณ์ที่เรียกว่า "truncation errors" รวมทั้ง errors แบบอื่นเกิดขึ้นเช่น "round-off errors" ปรากฏการณ์ที่เกิด errors ในหลายรูปแบบนี้จะมีความเกี่ยวข้องกับ time increment (h)

ตามปกติถ้าให้มีความแม่นยำดีขึ้น (better accuracy) ก็ควรเลือก Δt ที่มีค่าน้อย ($h = \Delta t$) ซึ่งก็มีผลโดยตรงต่อการคำนวณที่ต้องเพิ่มขึ้น จึงมีการตั้ง safe rule สำหรับวิธี finite difference นี้ ก็คือเลือกให้ $h \leq \tau_n/10$ ในกรณีที่ τ_n เป็นคาบธรรมชาติ (natural period) ของระบบ

4.1.2 การใช้ Finite Difference สำหรับ Damped System

เมื่อมี damping เกิดขึ้น สมการเชิง differential จะมีพจน์เพิ่มขึ้นอีก 1 ตัว คือพจน์ $\dot{x}_i$ ทำให้ได้ recurrence formula เป็น

$$x_{i+1} = 2x_i - x_{i-1} + h^2 f(x_i, \dot{x}_i, t_i) \qquad i \geq 2 \tag{38}$$

จากการที่เป็น damped system จึงจำเป็นต้องหาตัวแปรที่ไม่พึ่งพิงเช่นค่าของการเปลี่ยนตำแหน่ง (displacement) กับค่าของความเร็ว (velocity)

จาก Taylor series ของสมการ 33 หน้า 22 เมื่อพิจารณาถึงพจน์ 3 พจน์แรกของ series ทำให้ได้หาค่า x_1 ได้จาก expansion ของ x_{i+1} เมื่อ i = 0

$$x_1 = x_0 + \dot{x}_0 h + \frac{h^2}{2} f(x_0, \dot{x}_0, t_0) \tag{39}$$

ค่า quantity ของ $\dot{x}_1$ สามารถหาจากสมการชุดสอง (ของสมการ 33) สำหรับ x_{i-1} ด้วยค่า i = 1 จึงได้

$$x_0 = x_1 - \dot{x}_1 h + \frac{h^2}{2} f(x_0, \dot{x}_1, t_i) \tag{40}$$

ในทำนองเดียวกัน ค่า quantity ของ x_2 ที่ตั้งต้นคำนวณจากสมการ 38 ซึ่งกระบวนการก็จะซ้ำกับการหาค่าอื่นของ x_i และ x_i โดยอาศัย Taylor series ที่ได้แสดงวิธีการหาไว้ก่อนแล้วใน undamped system

4.2 วิธี Runge-Kutta

การวิเคราะห์ทาง numerical method ที่นำมาแก้ปัญหาของ second-order differentiation equation มีได้หลายรูปแบบ แต่ที่นิยมใช้มากก็จะเป็นกระบวนการคำนวณของ Runge-Kutta เพราะเหตุว่าวิธีการของ Runge-Kutta นี้เป็นการเริ่มต้นคำนวณด้วยตนเองและให้ผลที่มีระดับความแม่นยำดี โดยทั่วไปวิธีของ Runge-Kutta นี้จะใกล้เคียงมากกับ Taylor series expansion (ที่ใช้ในวิธี finite difference ในหัวข้อ 4.1)

การเริ่มต้นการวิเคราะห์ทาง numerical ของ Runge-Kutta จะลด order ของ second-order differential equations จำนวน 2 สมการ

ตัวอย่างที่นำมาใช้แก้ปัญหาทางด้านการคาดคะเนการสั่นสะเทือนของ dynamical differential equation ของระบบ SDF สามารถจะเขียนได้เป็น

$$\ddot{x} = \frac{1}{m}[f(t) - kx - c\dot{x}] = F(x, \dot{x}, t) \tag{41}$$

เมื่อทำการลด second-order equation ลงเหลือเป็น two first-order equations ก่อนทำการ integration ดังนั้นสมการในระบบเชิงพลศาสตร์จะเป็น

$$\left.\begin{aligned} \dot{x} &= y \\ \dot{y} &= f(x, y, t) \end{aligned}\right\} \tag{42}$$

จากสมการ 42 ข้างบนนี้ นำมาเขียนใหม่ได้เป็น

$$\left.\begin{aligned} \dot{x} &= y \\ \dot{y} &= F(x, y, t) \end{aligned}\right\} \tag{43}$$

ทั้งตัวแปร x และตัวแปร y จัดอยู่ในกลุ่มที่ใกล้เคียงของ x_i และ y_i เมื่อเป็นเช่นนี้ก็สามารถจะเขียนให้อยู่ในพจน์ของ Taylor series ถ้าหากให้มีช่วงการเพิ่มของเวลา (time increment, h)

มีค่าเท่ากับ Δt สามารถเขียน series expansion ได้คือ

$$\left.\begin{aligned} x &= x_i + \left(\frac{dx}{dt}\right)_i h + \left(\frac{d^2x}{dt^2}\right)_i \frac{h^2}{2} + \cdots \\ y &= y_i + \left(\frac{dy}{dt}\right)_i h + \left(\frac{d^2y}{dt^2}\right)_i \frac{h^2}{2} + \cdots \end{aligned}\right\} \tag{44}$$

อย่างไรก็ตามแทนที่จะแสดง expressions ของ series แบบสมการ 44 ก็ยังสามารถจะแทนค่าของ first derivative โดยค่าเฉลี่ยของความลาดเอียง (slope) และละทิ้งค่า higher order derivatives (เพราะมีค่าน้อย) ถ้าให้ av เป็นสัญกรณ์แทนคำ average ทำให้เขียนสมการที่ 44 เป็น

$$\left.\begin{aligned} x &= x_i + \left(\frac{dx}{dt}\right)_{iav} h \\ y &= y_i + \left(\frac{dy}{dt}\right)_{iav} h \end{aligned}\right\} \tag{45}$$

ถ้าหากต้องการให้เกิดการ fitting a second-order polynomial ก็อาจใช้ Simpson's rule [ซึ่ง modified มาจาก simple trapezoidal rule เพื่อที่จะทำการ appraximate curve] ซึ่งหาค่าเฉลี่ยของความลาดเอียงในช่วงระยะ h ทำให้ได้ค่า average slope เป็น

$$\left(\frac{dy}{dt}\right)_{iav} = \frac{1}{6}\left[\left(\frac{dy}{dt}\right)_{t_i} + 4\left(\frac{dy}{dt}\right)_{t_i+h/2} + \left(\frac{dy}{dt}\right)_{t_i+h}\right] \tag{46}$$

จากนั้นก็จะทำการหาความแตกต่างของพจน์ที่อยู่ตรงกลาง โดยทำคล้ายๆกับวิธีที่ใช้ Finite Difference แต่วิธี Runge-Kutta จะแยกพจน์กลาง (center term) ของสมการที่กำหนดให้แยกออกเป็น 2 พจน์และ 4 ค่า (ได้แก่ค่าของ t, x, y และ f) โดยถูกนำมาคำนวณสำหรับแต่ละจุดของ i ดังแสดงเป็นตารางที่ 1 ข้างล่างนี้

ตารางที่ 1 การคำนวณ center term แต่ละจุดของ i

t	x	$y = \dot{x}$	$f = \dot{y} = \ddot{x}$
$T_0 = t_i$	$X_0 = x_i$	$Y_0 = y_i$	$F_0 = f(T_0, X_0, Y_0)$
$T_1 = t_i + \frac{h}{2}$	$X_1 = x_i + Y_0\frac{h}{2}$	$Y_1 = y_i + F_0\frac{h}{2}$	$F_1 = f(T_1, X_1, Y_1)$
$T_2 = t_i + \frac{h}{2}$	$X = x_i + Y_1\frac{h}{2}$	$Y_3 = y_i + F_1\frac{h}{2}$	$F_2 = f(T_2, X_2, Y_2)$
$T_3 = t_i + h$	$X_3 = x_i + Y_2 h$	$Y_3 = y_i + F_2 h$	$F_3 = f(T_3, X_3, Y_3)$

เมื่อนำค่า quantities ของตารางที่ 1 ลงใน recurrence formula จะได้

$$x_{i+1} = x_i + \frac{h}{6}(Y_1 + 2Y_2 + 2Y_3 + Y_4) \tag{47}$$

$$y_{i+1} = y_i + \frac{h}{6}(F_1 + 2F_2 + 2F_3 + F_4) \tag{48}$$

เมื่อวิเคราะห์สมการทั้ง 2 ข้างบนนี้ จะเห็นได้ว่ามีค่า Y จำนวน 4 ค่า ซึ่งเมื่อหารด้วย 6 จะเป็นค่าเฉลี่ยของความลาดเอียง "x" [average slope "x" : (dx/dt)] และยังมีค่า F จำนวน 4 ค่าที่เมื่อหารด้วย 6 จะเป็นค่าเฉลี่ยของความลาดเอียง "y" [average slope "y" : (dy/dt)] ตามที่ได้แสดงไว้ในสมการ 45 หน้า 25

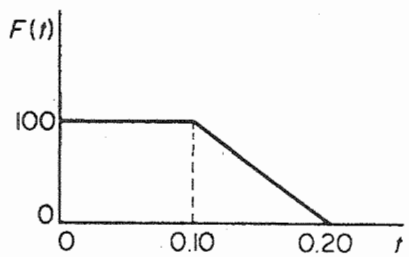
ตัวอย่างการคำนวณด้วยวิธี Runge-Kutta ในเรื่อง differential equation of motion คณะผู้วิจัยได้นำมาแสดง เพื่อแสดงถึงการเปรียบเทียบผลการคำนวณวิธี Runge-Kutta กับวิธี Finite Difference เพื่อเปรียบเทียบกับวิธี Exact Analytical Solution ทั้งนี้ต้องการแสดงผลของการหาค่าการเปลี่ยนตำแหน่งของอนุภาค (particle displacement) กับหาค่าความเร็วของอนุภาค (particle velocity) เมื่อเทียบกับเวลาของ real time (t)

1. <u>โจทย์ตัวอย่าง ของ SDF</u>

ก. โจทย์ข้อ 1 กำหนดให้ differential equation of motion เป็น

$$4\ddot{x} + 2{,}000x = F(t)$$

กำหนดให้ forcing function ดังรูปที่ 9 มีสภาวะเริ่มต้น $x_1 = \dot{x}_1 = 0$ (เหตุผลที่เลือก subscript เลข 1 ก็เพื่อให้สอดคล้องกับ t = 0 เพราะว่าในภาษาคอมพิวเตอร์ส่วนใหญ่ไม่สามารถมี subzero ได้)



รูปที่ 9 ฟังก์ชันของแรงภายนอกเมื่อเทียบกับเวลาสำหรับโจทย์ข้อ 1

Solution

หาค่าคาบธรรมชาติของระบบ ได้เป็น

$$\omega = \frac{2\pi}{\tau_n} = \sqrt{\frac{2000}{4}} = 22.36 \text{ rad/s}$$

$$\tau_n = \frac{2\pi}{22.36} = 0.281 \text{ s}$$

ตามวิธีการแรกทาง Finite Difference กำหนดว่า เพื่อป้องกัน truncation errors ที่เกี่ยวข้องกับการเพิ่มของเวลา h = Δt มี safe rule ที่ควรเลือก นั่นคือ เลือกให้ h $\leq \tau_n/10$ ดังนั้น เพื่อความสะดวกจึงเลือกให้ $h = 0.020$s ในการแทนค่าฟังก์ชัน F(t)

เมื่อทำการเขียนสมการการเคลื่อนที่ใหม่จะได้

$$\ddot{x} = \tfrac{1}{4}f(t) - 500x$$

ถ้าหากให้ y = $\dot{x}$ จะได้

$$\dot{y} = F(x,t) = \tfrac{1}{4}f(t) - 500x$$

เมื่อได้กำหนดให้ h = 0.02 จึงทำการคำนวณและใส่ตัวเลขเป็นตารางที่ 2

ตารางที่ 2 การคำนวณค่าตัวแปรเมื่อเทียบกับเวลา t_1 และ t_2

	t	x	$y = \dot{x}$	f
t_1 =	0	0	0	25
	0.01	0	0.25	25
	0.01	0.0025	0.25	23.75
t_2 =	0.02	0.0050	0.475	22.50

จากค่าต่างๆ ในตารางที่ 2 ข้างบนนี้จึงทำการหาค่าของ x_2 และ y_2 ได้ดังนี้

$$x_2 = 0 + \frac{0.02}{6}(0 + 0.50 + 0.50 + 0.475) = 0.00491667$$

$$y_2 = 0 + \frac{0.02}{6}(25 + 50 + 47.50 + 22.50) = 0.4833333$$

ในทำนองเดียวกันเมื่อต้องทำต่อถึงจุดที่ 3 ก็มีวิธีการทำซ้ำแบบเดิม เป็นตารางที่ 3

ตารางที่ 3 การคำนวณค่าตัวแปรเมื่อเทียบกับเวลา t_2 และ t_3

t_2 =	0.02	0.00491667	0.4833333	22.541665
	0.03	0.0097500	0.70874997	20.12500
	0.03	0.01200417	0.6845833	18.997915
t_3 =	0.04	0.01860834	0.8632913	15.695830

จึงหาคำนวณค่า x_3 และ y_3 ได้ดังนี้

$$x_3 = 0.00491667 + \frac{0.02}{6}(0.483333 + 1.4174999 + 1.3691666 + 0.8632913)$$
$$= 0.00491667 + 0.01377764 = 0.01869431$$
$$y_3 = 0.483333 + 0.38827775 = 0.87161075$$

จากนั้นก็สามารถทำการคำนวณหาค่าจุดต่างๆ อีกหลายจุดเพื่อให้สามารถพลอตเป็นกราฟด้วยวิธีการอย่างเดียวกัน สำหรับโจทย์ข้อ 1 ที่กำหนดให้นี้ ถ้าหากต้องการสมการการเคลื่อนที่ด้วยวิธีอื่นที่ไม่ใช่วิธี Runge-Kutta ก็สามารถทำได้

ตัวอย่างค่าการคำนวณของโจทย์ข้อ 1 นี้ได้สามารถทำไว้ 3 วิธี (ในตารางที่ 4) ได้แก่

ก. วิธีการแก้ปัญหาทาง analytical solution จึงได้ผลของค่าที่คำนวณเป็น exact solution

ข. วิธีการแก้ปัญหาทาง finite difference จึงได้ผลของค่าที่คำนวณเป็น central difference

ค. วิธีการแก้ปัญหาทาง numerical analysis ด้วยวิธีการ Runge-Kutta จึงได้ผลของค่าที่คำนวณแบบของ Runge-Kutta

จากตารางที่ 4 หน้าถัดไป ได้แสดงค่า x_1, x_2 ... x_n จะเห็นว่าเมื่อเปรียบเทียบผลของการคำนวณจะเห็นได้ว่าวิธี Runge-Kutta มีผลใกล้เคียงกับ exact solution มากกว่าวิธี Finite Difference ถึงแม้ว่าวิธี Runge-Kutta ไม่ต้องการประเมิน derivative สูงกว่าของ first derivative แต่ความถูกต้องแม่นยำเกิดได้เนื่องจากการประเมินค่า 4 ค่า ของ first derivatives ซึ่งสอดคล้องกับ

solution ของ Taylor series จนตลอดถึง order h^4 นอกจากนี้วิธีของ Runge-Kutta ยังสามารถแทนค่าตัวแปรแล้วแก้ปัญหาด้วย vector method ได้ด้วย

ตารางที่ 4 ค่าการคำนวณของ x_1 ถึง x_n เมื่อเทียบกับเวลา

Time t	*Exact Solution*	*Central Difference*	*Runge–Kutta*
0	0	0	0
0.02	0.00492	0.00503	0.00492
0.04	0.01870	0.01900	0.01869
0.06	0.03864	0.03920	0.03862
0.08	0.06082	0.06159	0.06076
0.10	0.08086	0.08167	0.08083
0.12	0.09451	0.09541	0.09447
0.14	0.09743	0.09807	0.09741
0.16	0.08710	0.08712	0.08709
0.18	0.06356	0.06274	0.06359
0.20	0.02949	0.02782	0.02956
0.22	−0.01005	−0.01267	−0.00955
0.24	−0.04761	−0.05063	−0.04750
0.26	−0.07581	−0.07846	−0.07571
0.28	−0.08910	−0.09059	−0.08903
0.30	−0.08486	−0.08461	−0.08485
0.32	−0.06393	−0.06171	−0.06400
0.34	−0.03043	−0.02646	−0.03056
0.36	0.00906	0.01407	0.00887
0.38	0.04677	0.05180	0.04656
0.40	0.07528	0.07916	0.07509
0.42	0.08898	0.09069	0.08886
0.44	0.08518	0.08409	0.08516
0.46	0.06463	0.06066	0.06473
0.48	0.03136	0.02511	0.03157

บทที่ 5

การวิเคราะห์คลื่นเชิง NUMERICAL ของระบบ N-DOF

สิ่งที่ปรากฏในระบบ Single Degree of Freedom ก็คือเมื่อระบบถูกกระตุ้นจะเกิดการไหวสะเทือนที่ความถี่ธรรมชาติของระบบ แต่ในระบบ Multidegree of Freedom ค่าของการสั่นสะเทือนตามธรรมชาติจะประยุกต์ใช้กับค่าตัวแปรเชิงอิลาสติก ที่สมมุติให้ระบบทั้งหมดเกิดการเคลื่อนที่

5.1 ธรรมชาติของระบบ Multidegree of Freedom

ถ้าหากระบบต้องการจุดพิกัดมากกว่าหนึ่ง เพื่ออธิบายการเคลื่อนที่ (motion) จะเรียกระบบนั้นว่า a multi-DOF system หรือ an N-DOF system ซึ่งค่า N คือจำนวนของจุดพิกัดที่ต้องการ ดังนั้น 2-DOF system ก็ต้องการจุดพิกัดที่ไม่พึ่งพิงกัน 2 จุด เพื่ออธิบายการเคลื่อนที่

ระบบ N-DOF ต่างจากระบบ SDF ตรงที่ค่าความถี่ธรรมชาติ N และแต่ละค่าของความถี่ธรรมชาติจะมีความสัมพันธ์กับการสั่นสะเทือนตามธรรมชาติ ซึ่งมีรูปแบบของการเปลี่ยนตำแหน่งเรียกว่า normal mode เทอมทางคณิตศาสตร์ที่สัมพันธ์กับค่าตัวแปรเหล่านี้เรียกว่า eigenvalues (characteristic values) และ eigenvectors (characteristic vectors)

ค่า eigenvalues และค่า eigenvectors เกิดมาจากสมการ N ของการเคลื่อนที่ที่อยู่ในเวลาเดียวกันของระบบ จึงเกี่ยวข้องกับ dynamic properties ของระบบ ในกรณีของค่า eigenvectors บางทีก็หมายถึง modal vectors และใช้แทนที่ลักษณะเชิงกายภาพของ natural modes กระบวนการที่จะปรับค่า elements ของ natural modes ให้ได้ค่าแอมพลิจูดเดียว (unique amplitude) จะมีชื่อเรียกว่า normalization เวคเตอร์ผลลัพธ์ที่ได้จาก normalization เรียกว่า normal modes

5.2 การวิเคราะห์การสั่นสะเทือนของ Normal Modes

การสั่นสะเทือนของ normal modes เป็นการสั่นสะเทือนของ free undamped vibrations ซึ่งขึ้นอยู่กับค่า mass กับค่า stiffness ของระบบ เมื่อเกิดการสั่นสะเทือนที่โหมดใดโหมดหนึ่งทุกจุดในระบบจะเกิดการเคลื่อนที่แบบ simple harmonic ผ่านตำแหน่งจุดสมดุลในทันทีทันใด

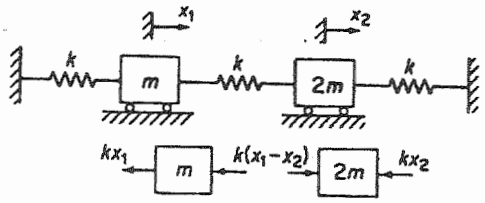
ถ้าหากจะทำให้เกิดการสั่นสะเทือนที่จุดหนึ่งของโหมด ระบบจะต้องมีสภาวะตั้งต้นที่สัมพันธ์กับ normal mode การวิเคราะห์ระบบที่มี Degree of Freedom มากกว่าหนึ่ง วิธีการเชิง

numerical ควรนำมาใช้โดยเฉพาะระบบที่มี degree สูงๆ วิธีเชิง matrix เป็นสิ่งจำเป็น ตัวอย่างดังต่อไปนี้จะนำไปสู่การศึกษาแบบจำลองที่มีค่า Degree of Freedom มากกว่าหนึ่งที่ต้องใช้คณิตศาสตร์ชั้นสูง

5.2.1 โจทย์ตัวอย่าง 2-DOF สำหรับ Normal Modes

ก. โจทย์ข้อ 1 ในเรื่อง Translational System

จากรูปที่ 10 ข้างล่างนี้เป็นระบบ undamped ที่มีพารามิเตอร์ที่เฉพาะเจาะจงของ 2-DOF จุดพิกัด x_1 และ x_2 วัดมาจากตำแหน่งของ inertial reference



รูปที่ 10 Free-body diagram ของมวลที่มีการออสซิเลชันในระบบ 2-DOF แบบ translational

Solution

จากไดอะแกรมของมวลทั้งสองนำไปสู่สมการการเคลื่อนที่ในรูปแบบของ differential equation

$$\left.\begin{aligned} m\ddot{x}_1 &= -kx_1 + k(x_2 - x_1) \\ 2m\ddot{x}_2 &= -k(x_2 - x_1) - kx_2 \end{aligned}\right\} \qquad (49)$$

ในการออสซิเลตของ normal mode มวลแต่ละอันจะมีการเคลื่อนที่แบบฮาร์โมนิคภายใต้ความถี่เดียวกันและผ่านจุดสมดุลอย่างทันทีทันใด ทำให้ได้ลักษณะการเคลื่อนที่คือ

$$\left.\begin{aligned} x_1 &= A_1 \sin \omega t \quad \text{or} \quad A_1 e^{i\omega t} \\ x_2 &= A_2 \sin \omega t \quad \text{or} \quad A_2 e^{i\omega t} \end{aligned}\right\} \qquad (50)$$

แทนค่าของ x_1 และ x_2 ลงในสมการดิฟเฟอเรนเชียลจะได้

$$\left.\begin{aligned}(2k-\omega^2 m)A_1 - kA_2 &= 0\\ -kA_1 + (2k-2\omega^2 m)A_2 &= 0\end{aligned}\right\} \tag{51}$$

ในกรณีที่จะ satisfy สำหรับค่าใดๆของ A_1 และ A_2 ค่า determinant ของสมการที่ 51 จะต้องเป็นศูนย์

$$\begin{vmatrix}(2k-\omega^2 m) & -k\\ -k & (2k-2\omega^2 m)\end{vmatrix} = 0 \tag{52}$$

กำหนดให้ $\omega^2 = \lambda$ และทำการ multiplying out ผลลัพธ์ของค่า determinant ตัวเดิม ใน สมการพีชคณิตยกกำลังสอง มีชื่อเรียกว่า characteristic equation

$$\lambda^2 - \left(3\frac{k}{m}\right)\lambda + \frac{3}{2}\left(\frac{k}{m}\right)^2 = 0 \tag{53}$$

รากสมการทั้งสองซึ่งได้แก่ λ_1 และ λ_2 จัดว่าเป็นค่า eigenvalues ของระบบ

$$\left.\begin{aligned}\lambda_1 &= \left(\frac{3}{2} - \frac{1}{2}\sqrt{3}\right)\frac{k}{m} = 0.634\frac{k}{m}\\ \lambda_2 &= \left(\frac{3}{2} + \frac{1}{2}\sqrt{3}\right)\frac{k}{m} = 2.366\frac{k}{m}\end{aligned}\right\} \tag{54}$$

และได้คำตอบค่า ความถี่ธรรมชาติ (natural frequencies) ของระบบคือ

$$\omega_1 = \lambda_1^{1/2} = \sqrt{0.634\frac{k}{m}}$$

$$\omega_2 = \lambda_2^{1/2} = \sqrt{2.366\frac{k}{m}}$$

จากสมการที่ 51 สามารถหาอัตราส่วนของแอมพลิจูดได้เป็น

$$\frac{A_1}{A_2} = \frac{k}{2k-\omega^2 m} = \frac{2k-2\omega^2 m}{k} \tag{55}$$

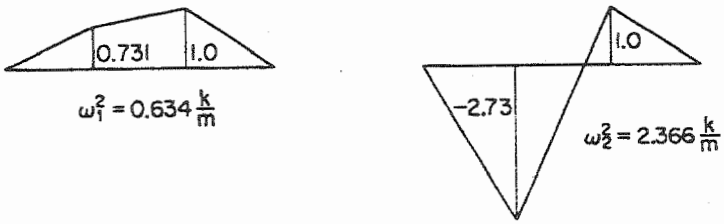
แทนค่าความถี่ธรรมชาติเหล่านี้ลงไปในสมการข้างบน คำตอบของอัตราส่วนของแอมพลิจูดที่มีค่าความถี่ธรรมชาติอันแรกเป็น $\omega_1^2 = 0.634$ k/m ได้แก่

$$\left(\frac{A_1}{A_2}\right)^{(1)} = \frac{k}{2k - \omega_1^2 m} = \frac{1}{2 - 0.634} = 0.731$$

สำหรับคำตอบของอัตราส่วนของแอมพลิจูดที่มีความถี่ธรรมชาติอันที่สองเป็น $\omega_2^2 = 2.366$ k/m ได้แก่

$$\left(\frac{A_1}{A_2}\right)^{(2)} = \frac{k}{2k - \omega_2^2 m} = \frac{1}{2 - 2.366} = -2.73$$

สมการที่ 55 หน้า 32 ช่วยให้สามารถหาอัตราส่วนของแอมพลิจูด ซึ่งเป็นค่า arbitrary ไม่ใช่ค่าสัมบูรณ์ (absolute) ถ้าหากต้องการจะเลือกกำหนดให้ค่าแอมพลิจูดหนึ่งมีค่าเท่ากับหนึ่ง (1) หรือมีค่าใดๆตามที่กำหนด ในเชิงคณิตศาสตร์จัดได้ว่าอัตราส่วนของแอมพลิจูดถูก normalized ไปยังจำนวณตัวเลขนั้น หรือเรียก normalized amplitude ratio ว่าเป็น normal mode ซึ่งใช้สัญลักษณ์เป็น $\varnothing_i(x)$



รูปที่ 11 แสดง normal modes ของระบบ 2-DOF ที่แสดงในรูปที่ 10

Normal modes ทั้งสองของโจทย์ตัวอย่างนี้ (ดูรูปที่ 11) มีชื่อเรียกว่า eigenvectors ซึ่งมีคำตอบของ eigenvectors ทั้งสองเป็น

$$\phi_1(x) = \begin{Bmatrix} 0.731 \\ 1.00 \end{Bmatrix} \qquad \phi_2(x) = \begin{Bmatrix} -2.73 \\ 1.00 \end{Bmatrix}$$

ถ้าหากจะเขียนการออสซิเลชันของแต่ละ normal mode จะแสดงได้ดังนี้

$$\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}^{(1)} = A_1 \begin{Bmatrix} 0.731 \\ 1.00 \end{Bmatrix} \sin(\omega_1 t + \psi_1)$$

$$\begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}^{(2)} = A_2 \begin{Bmatrix} -2.73 \\ 1.00 \end{Bmatrix} \sin(\omega_2 t + \psi_2)$$

Normal modes ที่แสดงเป็นแบบกราฟฟิก ดังรูปที่ 11 จะสังเกตได้ว่า ในกรณีของ normal mode อันแรกมวลทั้งสองจะมีการเคลื่อนที่แบบ in phase ไปในทางเดียวกัน ส่วนในกรณีของ normal mode อันที่สองมวลทั้งสองจะมีการเคลื่อนที่แบบ out of phase ที่เคลื่อนที่ไปในทางตรงกันข้าม

5.2.2 Modal Matrix สำหรับ N Normal Modes

ในขณะที่ N normal modes (หรือ ค่า eigenvectors) ถูกจัดให้เป็น square matrix ซึ่งแต่ละ normal mode จัดเป็น column หนึ่ง มีชื่อเรียกเฉพาะสำหรับ matrix แบบนี้ว่า Modal Matrix (P)

$$K\phi_i = \lambda_i M\phi_i \tag{56}$$

หลังจากการทำ transpose ของ mode j กับการคูณ matrix และการจัดรูปแบบใหม่จะได้ orthogonal charactor ของ normal modes ที่เกี่ยวข้องกับ symmetry mass matrix (M) เป็น

$$\phi_i^T M\phi_j = 0 \qquad i \neq j \tag{57}$$

และได้ orthogonal character ที่เกี่ยวข้อง symmetry stiffness matrix (K) เป็น

$$\phi_i^T K\phi_j = 0 \qquad i \neq j \tag{58}$$

ดังนั้น modal matrix สำหรับระบบ 3-DOF จะเขียนได้รูปแบบเป็น

$$P = \left[\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}^{(1)} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}^{(2)} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}^{(3)} \right] = [\phi_1 \phi_2 \phi_3] \tag{59}$$

เนื่องจาก natural modes of vibration มีคุณสมบัติที่สำคัญในตัวคือ orthogonality หรือกล่าวอีกนัยหนึ่งว่า normal modes มีลักษณะเชิง orthogonal กับ mass และ stiffness matrices

ถ้าหากใช้สัญกรณ์ $\varnothing_i$ สำหรับค่า eigenvector ที่ ith สมการ normal mode ของ ith mode คือ ค่าความสัมพันธ์ตั้งแต่สมการที่ 56 - 59 สามารถนำมาใช้ได้กับ modal matrix ของ สมการที่ 56 หน้าก่อนนี้ เพื่อทำให้ความสัมพันธ์เชิง orthogonality เป็นสมการเดียว จึงทำปฏิบัติการเชิงคณิตศาสตร์โดยการ transpose of P ได้เป็น

$$P^T = \begin{bmatrix} (x_1 x_2 x_3)^{(1)} \\ (x_1 x_2 x_3)^{(2)} \\ (x_1 x_2 x_3)^{(3)} \end{bmatrix} = [\phi_1 \phi_2 \phi_3]^T \tag{60}$$

สมการที่ 60 ข้างบนนี้ แต่ละแถวจะเป็นโหมดแต่ละโหมด ถ้าหากฟอร์ม product $P^T KP$ ผลลัพธ์จะเป็น diagonal matrix มีค่าเท่ากับศูนย์

ตัวอย่างในระบบ 3-DOF สามารถทำปฏิบัติการด้วย modal matrix ต่อ mass ได้รูปแบบของ mass matrix เป็น

$$\begin{aligned} P^T MP &= [\phi_1 \phi_2 \phi_3]^T [M][\phi_1 \phi_2 \phi_3] \\ &= \begin{bmatrix} \phi_1^T M\phi_1 & \phi_1^T M\phi_2 & \phi_1^T M\phi_3 \\ \phi_2^T M\phi_1 & \phi_2^T M\phi_2 & \phi_2^T M\phi_3 \\ \phi_3^T M\phi_1 & \phi_3^T M\phi_2 & \phi_3^T M\phi_3 \end{bmatrix} = \begin{bmatrix} M_{11} & 0 & 0 \\ 0 & M_{22} & 0 \\ 0 & 0 & M_{33} \end{bmatrix} \end{aligned} \tag{61}$$

ในสมการที่ 61 ข้างบน เทอมของ off-diagonal มีค่าเป็นศูนย์เนื่องจากเป็น orthogonality และเทอม diagonal เป็น generalized mass, M_{ii}

ในการทำ formulation ที่คล้ายคลึงกันจะได้ค่า stiffness จากการทำปฏิบัติการดังนี้

$$P^T KP = \begin{bmatrix} K_{11} & 0 & 0 \\ 0 & K_{22} & 0 \\ 0 & 0 & K_{33} \end{bmatrix} \tag{62}$$

เทอมของ diagonal เป็น generalized stiffness, K_{ii}

เมื่อค่า normal modes, $\varnothing_i$ ใน P matrix ถูกแทนที่โดย orthonormal modes, $\tilde{\phi}_i$ ก็จะแทนสัญกรณ์ของ modal matrix ด้วย $\tilde{P}$ ได้ความสัมพันธ์เชิง orthogonality เป็น

$$\tilde{P}^T M \tilde{P} = I \tag{63}$$

$$\tilde{P}^T K \tilde{P} = \Lambda \tag{64}$$

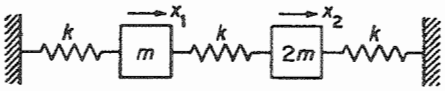
ซึ่งค่า Λ เป็น diagonal matrix ของ eigenvalues

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \tag{65}$$

5.2.3 โจทย์ตัวอย่างของ 2-DOF เรื่อง Modal Matrix

โจทย์ข้อ 2

ใช้ตัวอย่างโจทย์ข้อที่ 1 ในหัวข้อ 5.2.1 ใช้ในการตกแต่งผลลัพธ์ของระบบ (ดูรูปที่ 12) โดยการแทนค่าลงไปในสมการต่างๆของ modal matrix (สมการที่ 59-65)



รูปที่ 12 ไดอะแกรมที่ปรับปรุงมาจากโจทย์ข้อที่ 1 ในหัวข้อ 5.2.1

Solution

จากไดอะแกรมข้างบนได้ค่าของ mass and stiffness matrices คือ

$$M = m\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \qquad K = k\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

จากโจทย์ข้อที่ 1 ได้ค่า eigenvectors เป็น

$$\lambda_1 = \frac{\omega_1^2 m}{k} = 0.634 \qquad \phi_1 = \begin{Bmatrix} 0.731 \\ 1.000 \end{Bmatrix}$$
$$\lambda_2 = \frac{\omega_2^2 m}{k} = 2.366 \qquad \phi_2 = \begin{Bmatrix} -2.73 \\ 1.00 \end{Bmatrix}$$

ค่าต่างๆ ข้างบนเมื่อจัดกลุ่มเป็น modal matrix, P คือ

$$P = \begin{bmatrix} 0.731 & -2.73 \\ 1.00 & 1.00 \end{bmatrix}$$
$$P^T M P = \begin{bmatrix} 0.731 & 1.0 \\ -2.73 & 1.0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 0.731 & -2.73 \\ 1 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 2.53 & 0 \\ 0 & 9.45 \end{bmatrix} = \begin{bmatrix} M_{11} & 0 \\ 0 & M_{22} \end{bmatrix}$$

อ่านค่าจาก modal matrix ข้างบนได้ค่า generalized mass เป็น 2.53 และ 9.45

ถ้าหากจะทำ cross check แทนที่จะหา modal matrix, P เราใช้ orthonormal modes แทน จะได้

$$\tilde{P} = \left[\frac{1}{\sqrt{2.53}} \begin{Bmatrix} 0.731 \\ 1.00 \end{Bmatrix} \frac{1}{\sqrt{9.45}} \begin{Bmatrix} -2.73 \\ 1.00 \end{Bmatrix} \right] = \begin{bmatrix} 0.459 & -0.888 \\ 0.628 & 0.3253 \end{bmatrix}$$
$$\tilde{P}^T M \tilde{P} = \begin{bmatrix} 0.459 & 0.628 \\ -0.888 & 0.325 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 0.459 & -0.888 \\ 0.628 & 0.325 \end{bmatrix} = \begin{bmatrix} 1.00 & 0 \\ 0 & 1.00 \end{bmatrix}$$
$$\tilde{P}^T K \tilde{P} = \begin{bmatrix} 0.459 & 0.628 \\ -0.888 & 0.325 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 0.459 & -0.888 \\ 0.628 & 0.325 \end{bmatrix}$$
$$= \begin{bmatrix} 0.635 & 0 \\ 0 & 2.365 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

ดังนั้นเทอม diagonal จะได้ค่าใกล้เคียงค่า eigenvalues ของตัวอย่างโจทย์ข้อที่ 1

5.3 วิธีการคำนวณของระบบ N-DOF

จากหัวข้อ 5.1 - 5.2 ได้กล่าวถึงกระบวนการพื้นฐานในการหาค่า eigenvalues และ eigenvectors ของระบบโดยการหารากของ polynomial equation ที่ได้มาจากค่า characteristic determinant จากนั้นนำค่าของราก (หรือค่า eigenvalues) มาแทนที่ทีละตัวเข้าไปในสมการเคลื่อนที่เพื่อหารูปร่างของโหมด (หรือค่า eigenvectors) ของระบบ

ในอีกแนวทางหนึ่ง สามารถที่จะใช้วิธีการทางคณิตศาสตร์ที่มีการ transformation ของจุดพิกัดควบคู่กับ iteration procedure ทำให้ได้ผลลัพธ์ของค่า eigenvalues และ eigenvectors พร้อมกันในเวลาเดียวกัน

วิธีการแนวใหม่นี้ มีสมการของการเคลื่อนที่เป็น

$$[-\lambda M + K]X = 0 \tag{66}$$

สมการการเคลื่อนที่ข้างบนจำต้องแปรเปลี่ยนเป็นรูปแบบค่ามาตรฐานของ eigenvalues ที่ต้องนำไปใช้ในโปรแกรมคอมพิวเตอร์ส่วนใหญ่ รูปแบบมาตรฐานได้แก่

$$[\bar{A} - \lambda I]Y = 0 \tag{67}$$

โดยที่ $\bar{A}$ = square symmetric matrix

Y = new displacement vector transformed from X

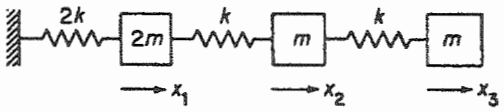
รูปแบบของสมการ 67 ข้างบนนี้ใช้ได้กับวิธีการ transformation และ matrix iteration

5.3.1 ตัวอย่างการแก้ปัญหารากของสมการในระบบ N-DOF

โจทย์ข้อที่ 3

ตัวอย่างโจทย์ข้อนี้เป็นวิธีการที่จะนำไปสู่การใช้คอมพิวเตอร์แก้ปัญหารากของสมการในระบบ 3-DOF

จากรูปที่ 13 ข้างล่างเป็นไดอะแกรมการเคลื่อนที่เชิง translational ในระบบ 3-DOF ที่ต้องการหาค่ารากของสมการ



รูปที่ 13 แสดงไดอะแกรมของระบบ 3-DOF ซึ่งมีค่า normal modes และ natural frequencies ตามต้องการ

Solution

จากรูปที่ 13 สามารถเขียนสมการการเคลื่อนที่ของระบบเป็น

$$m\begin{bmatrix}2 & & \\ & 1 & \\ & & 1\end{bmatrix}\begin{Bmatrix}\ddot{x}_1\\ \ddot{x}_2\\ \ddot{x}_3\end{Bmatrix}+k\begin{bmatrix}3 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 1\end{bmatrix}\begin{Bmatrix}x_1\\ x_2\\ x_3\end{Bmatrix}=\begin{Bmatrix}0\\ 0\\ 0\end{Bmatrix}$$

or

$$\left(-\lambda\begin{bmatrix}2 & & \\ & 1 & \\ & & 1\end{bmatrix}+\begin{bmatrix}3 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 1\end{bmatrix}\right)\begin{Bmatrix}x_1\\ x_2\\ x_3\end{Bmatrix}=\begin{Bmatrix}0\\ 0\\ 0\end{Bmatrix}$$

โดยที่ $\lambda = \omega^2 m/k$

ค่า eigenvalues ของระบบหาได้จาก characteristic determinant ที่ทำให้เท่ากับศูนย์

$$\begin{vmatrix} (3-2\lambda) & -1 & 0 \\ -1 & (2-\lambda) & -1 \\ 0 & -1 & (1-\lambda) \end{vmatrix} = 0$$

ค่า determinant นี้ เมื่อนำวิธี Minor ในกระบวนการปฏิบัติการของ matrix มาใช้ จะสามารถถูกลดให้เป็นสมการพีชคณิตยกกำลังสาม โดยการเลือกแถวแรกของ columns เป็น pivots

$$(3-2\lambda)\begin{vmatrix} (2-\lambda) & -1 \\ -1 & (1-\lambda) \end{vmatrix} + 1\begin{vmatrix} -1 & 0 \\ -1 & (1-\lambda) \end{vmatrix} = 0$$

ซึ่งทำให้ได้ characteristic equation เป็น

$$\lambda^3 - 4.50\lambda^2 + 5\lambda - 1 = 0$$

สมการ characteristic equation นี้เป็นปัญหาทางคณิตศาสตร์ที่ไม่มีวิธีหารากกำลังสามด้วยวิธีการง่ายๆ ซึ่งนำไปใช้ในโปรแกรมคอมพิวเตอร์ เพื่อหารากของสมการ (eigenvalues) ของ polynomial equation วิธีการนี้มีหลายขั้นตอนแต่ตรงไปตรงมา ดังนี้

ก. <u>ขั้นตอนที่ 1</u>

สมมุติค่า ฟังก์ชันที่สัมพันธ์กับ N-degree equation เป็น

$$f(\lambda) = \lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + \cdots + c_n = 0$$

ข. <u>ขั้นตอนที่ 2</u>

สมมุติตัวเลขสำหรับ λ และแทนค่า λ ลงไปในสมการก็จะได้ค่าสำหรับ $f(\lambda)$

ค. <u>ขั้นตอนที่ 3</u>

ถ้ากระบวนการมีการทำซ้ำกัน เมื่อทำการพลอตค่า $f(\lambda)$ เป็นฟังก์กับค่า λ ช่วงที่มีการเปลี่ยนเครื่องหมาย (zero crossing) ใน $f(\lambda)$ จะแสดงเป็นค่าประมาณของรากของสมการ

เมื่อนำวิธี interpolation แบบ straight line หรือ Newton's interpolation จะสามารถหารากของสมการได้

ง. <u>วิธีการที่ใช้ในโปรแกรมคอมพิวเตอร์</u>

ปกติจะใช้หาค่าประมาณของพิสัย (range) ที่ครอบคลุมโดย λ และช่วง (interval) ของ $\Delta\lambda$ โดยคอมพิวเตอร์ สมการ polynomial สามารถจะถูก factored ได้เป็น

$$f(\lambda) = (\lambda - \lambda_1)(\lambda - \lambda_2)(\lambda - \lambda_3)\cdots = 0$$

โดยที่ λ_i เป็นรากฐานของสมการ เมื่อทำการคูณและ factored ตัวสมการ ทำให้ค่าสัมประสิทธิ์ c_1 สำหรับค่ายกกำลังสูงสุดต่อไปของ λ จะมีค่าเท่ากับผลบวกของรากฐานทั้งหมดเสมอไป ไม่ว่าค่า N จะมีค่าอย่างไร นั่นคือ สำหรับสมการยกกำลังสาม จะได้

$$f(\lambda) = \lambda^3 - (\lambda_1 + \lambda_2 + \lambda_3)\lambda^2 + (\lambda_1\lambda_2 + \lambda_1\lambda_3 + \lambda_2\lambda_3)\lambda - \lambda_1\lambda_2\lambda_3 = 0$$

กระบวนการที่แสดงในหัวข้อ ง. นี้นำมาใช้หรือ modified ได้สำหรับโปรแกรมคอมพิวเตอร์เนื่องจากความสามารถของคอมพิวเตอร์ในการคำนวณค่า $\Delta\lambda$ เป็นพันๆค่าด้วยเวลาอันรวดเร็ว ทำให้การ interpolation ใกล้เคียงแม่นยำมาก

5.3.2 <u>Gauss Elimination</u>

วิธีของ Gauss ใช้แก้ปัญหารูปร่างของโหมด วิธีนี้เป็นแนวทางอีกแนวทางหนึ่ง ซึ่งจะหาอัตราส่วนของแอมพลิจูดที่แตกต่างจากเดิมที่ใช้แทนค่า eigenvalues ทีละครั้งในสมการของการเคลื่อนที่

ข้อดีของวิธีการของ Gauss คือ กระบวนการนี้จะลดสมการแมตริกซ์ไปเป็น upper triangular form ซึ่งสามารถจะแก้ปัญหาสำหรับค่าแอมพลิจูดที่ตั้งต้นจากส่วนล่างสุดของสมการแมตริกซ์ในการประยุกต์ใช้วิธีของ Gauss เริ่มต้นที่เขียนสมการการเคลื่อนที่ในเทอมของ λ

$$\begin{bmatrix} (3-2\lambda_i) & -1 & 0 \\ -1 & (2-\lambda_i) & -1 \\ 0 & -1 & (1-\lambda_i) \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}^{(i)} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

ค่า eigenvalues ที่ใช้ในการแก้ปัญหานี้ คือ

$$\lambda = \omega^2 \frac{m}{k} = \begin{Bmatrix} 0.25536 \\ 1.3554 \\ 2.8892 \end{Bmatrix}$$

แทนค่า $\lambda_1 = 0.25536$ เข้าในสมการอันก่อน จะได้

$$\begin{bmatrix} 2.489 & -1 & 0 \\ -1 & 1.745 & -1 \\ 0 & -1 & 0.7446 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}^{(1)} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

ก. <u>ขั้นตอนที่ 1</u>

ในวิธีของ Gauss ขั้นแรกจะจำกัดเทอมของคอลัมน์แรกให้ไปอยู่ในแถวที่ 2 และแถวที่ 3 เนื่องจากคอลัมน์แรกของแถวที่ 3 มีค่าเท่ากับศูนย์ จึงมีความจำเป็นเพียงแต่ทำให้เทอมแรกของแถวที่สองเป็นศูนย์

ข. <u>ขั้นตอนที่ 2</u>

เพื่อทำให้เทอมแรกของแถวที่สองเป็นศูนย์ ก็ทำการหาแถวแรกด้วย 2.489 และบวกเข้าไปในแถวที่สอง ซึ่งจะได้

$$\begin{bmatrix} 2.489 & -1 & 0 \\ 0 & 1.343 & -1 \\ 0 & -1 & 0.7446 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}^{(1)} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

ค. <u>ขั้นตอนที่ 3</u>

อาจไม่มีความจำเป็นต้องทำปฏิบัติการอะไรต่อไป โดยอาจทำกระบวนการซ้ำเดิมเพื่อจำกัดเทอม -1 ของแถวที่สาม ซึ่งทำได้โดยการหารแถวที่สองด้วย 1.343 และบวกเข้าไปในแถวที่สาม ซึ่งจะได้ผลลัพธ์เป็น

$$\begin{bmatrix} 2.489 & -1 & 0 \\ 0 & 1.343 & -1 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}^{(1)} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

ทั้งนี้ไม่ว่าจะเป็นสมการข้างบนนี้หรือสมการก่อนหน้านี้ ค่าแอมพลิจูด x_3 จะถูกกำหนดให้เป็น 1 ซึ่งเป็นผลลัพธ์ของ eigenvector ตัวแรกหรือโหมดแรก

$$\phi_1 = \begin{Bmatrix} x_1 \\ x_2 \\ 1 \end{Bmatrix}^{(1)} = \begin{Bmatrix} 0.2992 \\ 0.7446 \\ 1.000 \end{Bmatrix}$$

ง. <u>ขั้นตอนที่ 4</u>

โดยการทำปฏิบัติการด้วยกระบวนซ้ำๆกันสำหรับค่าของ λ_2 และ λ_3 ค่า eigenvectors สำหรับโหมดที่สองและโหมดที่สามก็สามารถหาได้

5.3.3 <u>Matrix Iteration</u>

วิธีการนี้เป็นวิธีการหาผลลัพธ์ของสมการพีชคณิตของ nth order ในระบบ N-DOF ที่แตกต่างจากวิธีการ expansion ของ determinant equation ประโยชน์ของ matrix iteration ช่วยในการ formulated สมการการเคลื่อนที่ในเรื่องของ flexibility matrix และ stiffness matrix

ในกระบวนการของ matrix iteration เทอมของ dynamic matrix, $\bar{A}$ ไม่จำเป็นต้องเป็น symmetry เมื่อกำหนดให้ค่าเทอมของ flexibility matrix [a] = K^{-1} จะทำให้ได้สมการของ normal mode vibration คือ

$$\bar{A}X = \bar{\lambda}X \tag{68}$$

โดยที่ $\bar{A} = [a][m] = K^{-1}M$

$\bar{\lambda} = 1/\omega^2$

ก. <u>ขั้นตอนที่ 1</u>

กระบวนการของ iteration เริ่มต้นโดยสมมุติให้มีเซทของแอมพลิจูดเซทหนึ่งสำหรับคอลัมน์ซ้ายของสมการ 68 และทำปฏิบัติการได้ผลลัพธ์ของตัวเลขเป็นคอลัมน์

ข. <u>ขั้นตอนที่ 2</u>

เป็นวิธีการ normalization ซึ่งทำให้ค่าแอมพลิจูดอันหนึ่งถูก normalized ให้เท่ากับค่า unity และทำการหารเทอมแต่ละเทอมของคอลัมน์ด้วยค่าแอมพลิจูดที่ถูก normalized แล้ว

ค. ขั้นตอนที่ 3

กระบวนการถูกทำให้ซ้ำๆกันด้วย normalized column จนกระทั่งค่าแอมพลิจูดมีเสถียรภาพที่ค่าของแพทเทิร์นที่แน่นอน จนกระทั่งค่า normalized column ไม่ได้แตกต่างจากค่าที่ได้จากการ iteration ครั้งก่อน

ค่าสุดท้ายที่ได้ คือ eigenvector ถูก converged ไปเป็นค่า eigenvalue ที่ใหญ่สุด หรือในกรณีนี้ทำให้ค่า natural frequency, ω_1 มีค่าต่ำ(เล็ก)สุด

5.3.4 Sweeping Matrix

สมการการเคลื่อนที่เมื่อถูก formulated เข้าไปในเทอมของ flexibility กระบวนการ iteration จะ converge ไปเป็น lowest mode ผลเสียของ iteration ก็คือ มีการนำค่า small component ของ mode shape, $\varnothing_1$ เข้าไปในแต่ละ iteration ทำให้เกิด round-off errors จึงจำเป็นต้องนำ component นี้ออกจากแต่ละ iterated vector เพื่อให้เกิด iteration ที่จะ converge ไปเป็น $\varnothing_2$

เพื่อให้กระบวนการย้าย component นี้ออกเสร็จสิ้นสมบรูณ์ จำเป็นต้องใช้ทฤษฎีของ Expansion Theorem

$$X = c_1\phi_1 + c_2\phi_2 + c_3\phi_3 + \cdots + \tag{69}$$

ต่อไปก็ทำการคูณสมการด้วย $\varnothing_1^{\mathrm{T}}\mathrm{M}$ โดยที่ค่า $\varnothing_1$ เป็น first normal mode สามารถหามาได้ คือ

$$\phi_1^T MX = c_1\phi_1^T M\phi_1 + c_2\phi_1^T M\phi_2 + c_3\phi_1^T M\phi_3 + \cdots + \tag{70}$$

เนื่องจากลักษณะเชิง orthogonality เทอมทางด้านขวามือของสมการ 71 จะมีค่าศูนย์ (ยกเว้นเทอมแรก)

$$\phi_1^T MX = c_1\phi_1^T M\phi_1 \tag{71}$$

จากสมการที่ 69 ถ้าหากค่า $c_1 = 0$ จะมีค่าการเปลี่ยนตำแหน่งปราศจาก $\varnothing_1$ ทั้งนี้เพราะว่า $\varnothing_1^T M \varnothing_1$ ไม่สามารถเป็นศูนย์ นั่นคือจะได้ constraint equation เป็น

$$\phi_1^T MX = 0 \tag{72}$$

เมื่อต้องการแก้ปัญหาของ dynamic matrix สำหรับ 3 x 3 problem จะได้เป็น

$$\phi_1^T MX = \left(x_1^{(1)} x_2^{(1)} x_3^{(1)}\right) \begin{bmatrix} m_1 & & \\ & m_2 & \\ & & m_3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}$$

$$= m_1 x_1^{(1)} x_1 + m_2 x_2^{(1)} x_2 + m_3 x_3^{(1)} x_3 = 0$$

โดยที่ $x_1^{(1)}$, $x_2^{(2)}$ และ $x_3^{(3)}$ เป็นตัวแปรที่ทราบค่าและค่า x_1 ที่มีค่า superscript เนื่องจากมีการ iterated ของเวคเตอร์ X ไปจนถึงที่ ith

$$x_1 = -\left(\frac{m_2}{m_1}\right)\left(\frac{x_2}{x_1}\right)^{(1)} x_2 - \left(\frac{m_3}{m_1}\right)\left(\frac{x_3}{x_1}\right)^{(1)} x_3$$

$$x_2 = x_2$$

$$x_3 = x_3$$

เมื่อทำปฏิบัติการต่อไปจะได้ matrix form สุดท้ายเป็น

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{bmatrix} 0 & -\left(\frac{m_2}{m_1}\right)\left(\frac{x_2}{x_1}\right)^{(1)} & -\left(\frac{m_3}{m_1}\right)\left(\frac{x_3}{x_1}\right)^{(1)} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} \tag{73}$$

$$= [S]\{X\}$$

สมการที่ 73 เป็น constraint equation ที่ใช้สำหรับการย้ายค่า first mode และค่า [S] เป็น sweeping matrix

เมื่อทำการแทนที่ X ใน dynamic martix สำหรับสมการ normal mode vibration ของสมการที่ 68 หัวข้อ 5.3.3 เมื่อแทนค่า X ทางด้านซ้ายของสมการ 73 ทำให้ constraint equation ของสมการ 73 เปลี่ยนเป็น

$$\bar{A}SX = \bar{\lambda}X \tag{74}$$

ในการ iteration ของสมการที่ 74 นี้ ได้จำกัดตัวแปรที่ไม่ต้องการออก (sweep out) ได้แก่ ค่า $\varnothing_1$ component ในแต่ละขั้นตอนของ iteration และถูก converge ไปเป็น second mode, $\varnothing_2$

สำหรับ third mode และ higher modes กระบวนการ sweeping จะถูกกระทำซ้ำจนกระทั่งได้ค่าของ normal mode วิธีการเช่นนี้จะทำให้เกิดการลด order ของ matrix equation ไปทีละหนึ่ง (1) ในแต่ละเวลาของการ sweeping ดังนั้นค่า matrix [AS] จะถูกเรียกว่าเป็น deflated matrix

5.3.5 พื้นฐานของโครงสร้างโปรแกรมในกระบวนการ Iteration

สมการเริ่มต้นของ Expansion Theorem :

$$X_1 = c_1\phi_1 + c_2\phi_2 + c_3\phi_3 + \cdots +$$

ในกระบวนการ iteration ที่จะ converge ไปเป็นค่า eigenvalue ใหญ่(สูง)สุด จุดเริ่มต้นก็ต้องทำการคูณโดย dynamic matrix, $\bar{A}$ ทำให้เป็น

$$\bar{A}X_1 = X_2 = c_1\bar{A}\phi_1 + c_2\bar{A}\phi_2 + c_3\bar{A}\phi_3 + \cdots + \tag{75}$$

เมื่อทำปฏิบัติการเพิ่มเติมเพื่อหา new displacement vector, X_2 จะได้ผลลัพธ์เป็น

$$X_2 = c_1\frac{1}{\omega_1^2}\phi_1 + c_2\frac{1}{\omega_2^2}\phi_2 + c_3\frac{1}{\omega_3^2}\phi_3 + \cdots +$$

หลังจากทำกระบวนการซ้ำๆหลายครั้ง ทำให้ได้สมการสุดท้ายในการ convergence เป็น

$$AX_{n-1} = X_n = c_1\frac{1}{\omega_1^{2n}}\phi_1 + c_2\frac{1}{\omega_2^{2n}}\phi_2 + c_3\frac{1}{\omega_3^{2n}}\phi_3 + \cdots + \tag{76}$$

ทั้งนี้เพราะว่า $\omega_n^2 > \omega_{n-1}^2 > \ldots\ldots > \omega_2^2 > \omega_1^2$ กระบวนการ convergence แปลงเป็น fundamental mode

ในการทำโปรแกรมของกระบวนการ iteration จำเป็นต้องเปลี่ยนแปลงรูปแบบ (form) ของ sweeping matrix, S การ transformation สำหรับเวคเตอร์แมตริกซ์ใช้วิธีการของ Gram-Schmidt orthogonalization ทำให้เขียนสมการที่ 75 ได้ใหม่เป็น

$$X_2 = X_1 - \alpha_1\phi_1 = c_2\phi_2 + c_3\phi_3 + \cdots + \tag{77}$$

ในเทอมของ $c_1\varnothing_1$ นั้นค่า component ของ $\varnothing_1$ เป็นค่าที่ไม่ต้องการ จึงต้องคูณสมการที่ 77 โดยใช้เทอมของ $\varnothing_1^T M$ ทำให้ได้เทอมเท่ากับศูนย์

$$\phi_1^T M(X_1 - \alpha\phi_1) = 0$$

ได้ค่าคงที่ของ α_1 เป็น

$$\alpha_1 = \frac{\phi_1^T M X_1}{\phi_1^T M \phi_1} \tag{78}$$

แทนค่า α_1 ในสมการที่ 77 และจัดเทอมใหม่จะได้

$$X_2 = X_1 - \frac{\phi_1\phi_1^T M}{\phi_1^T M\phi_1}X_1 = \left[1 - \frac{\phi_1\phi_1^T M}{\phi_1^T M\phi_1}\right]X_1$$

ดังนั้นจะได้ค่า expression สำหรับ sweeping matrix ที่เหมาะสำหรับไปใช้ในโปรแกรมคอมพิวเตอร์ คือ

$$S = \left[1 - \frac{\phi_1\phi_1^T M}{\phi_1^T M\phi_1}\right] \tag{79}$$

5.3.6 การ Transformation ของจุดพิกัดในรูปแบบมาตรฐานของฟอร์มคอมพิวเตอร์

สมการสำหรับ normal mode vibration ตามปกติจะเขียนเป็นแบบของ

$$[-\lambda M + K]X = 0 \tag{80}$$

ซึ่งค่า M และ K เป็น square symmetric matrix ทั้งคู่ และ λ เป็นค่า eigenvalue ที่มีความสัมพันธ์ต่อความถี่ธรรมชาติโดย $\lambda = \omega^2$ เมื่อคูณสมการที่ 80 ด้วย M^{-1} จะได้

$$[-\lambda I + A]X = 0 \tag{81}$$

ค่า A เป็น dynamic matrix โดยที่ $A = M^{-1} K$ ซึ่งตามปกติเทอม $M^{-1} K$ ไม่ symmetric ถ้าคูณสมการ 80 ด้วย K^{-1} จะได้

$$[\bar{A} - \bar{\lambda} I]X = 0 \tag{82}$$

ค่า $\bar{A}$ เป็น dynamic matrix ที่ไม่ symmetric โดยที่ $\bar{A} = K^{-1} M$ และ $\bar{\lambda}$ เป็น eigenvalue สำหรับสมการ และ $\bar{\lambda} = 1/\omega^2 = 1/\lambda$

เมื่อต้องการทำให้เป็นฟอร์มมาตรฐานสำหรับใช้ในคอมพิวเตอร์จำเป็นต้องมีการ transformation ของจุดพิกัด

$$X = U^{-1}Y \tag{83}$$

จากนั้นก็นำค่า X ไปใส่ในสมการที่ 80 และทำการ transpose เทอม U^{-1} จะได้

$$[-\lambda U^{-T} M U^{-1} + U^{-T} K U^{-1}]Y = 0 \tag{84}$$

ถ้าหากทำการ decompose ไม่ว่าเทอมของ M หรือ K เข้าไปในเทอม $U^T U$ ในสมการที่ 84 จะได้ฟอร์มมาตรฐานของสมการการเคลื่อนที่

นั่นคือ ถ้าให้ $M = U^T U$ สมการที่ 84 จะกลายเป็น

$$[-\lambda I + U^{-T} K U^{-1}]Y = 0 \qquad \lambda = \omega^2 \tag{85}$$

หรือถ้าให้ K = $U^T U$ สมการที่ 84 จะกลายเป็น

$$[U^{-T} M U^{-1} - \bar{\lambda} I]Y = 0 \qquad \bar{\lambda} = 1/\omega^2 \tag{86}$$

สมการที่ 85 และ 86 สามารถเขียนเป็นฟอร์มมาตรฐานเป็น

$$[-\lambda I + \tilde{A}]Y = 0$$

ซึ่ง dynamic matrix $\tilde{A}$ เป็น symmetric

5.3.7 Choleski Decomposition

ในกรณีที่ matrix M หรือ matrix K เต็ม ค่าของ matrix U และ matrix U^{-1} สามารถหาได้จาก Choleski decomposition ซึ่งทำได้ง่ายๆโดยเขียนสมการ M = $U^T U$ (หรือ K = $U^T U$) ให้อยู่ในเทอมของ upper triangular matrix สำหรับ U และ transpose ของ U คือ

$$\underset{U^T}{\begin{bmatrix} u_{11} & 0 & 0 \\ u_{12} & u_{22} & 0 \\ u_{13} & u_{23} & u_{33} \end{bmatrix}} \underset{U}{\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}} = \underset{M}{\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}} \tag{87}$$

โดยการคูณทางด้านซ้ายมือและทำให้เทอมทางด้านขวามือเท่ากัน ค่า u_{ij} แต่ละค่าสามารถหาจากเทอมของสัมประสิทธิ์ทางด้านขวา

ค่า inverse ของ upper triangular matrix ก็สามารถหามาได้จากสมการ UU^{-1} = I

$$\underset{U}{\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}} \underset{U^{-1}}{\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ 0 & b_{22} & b_{23} \\ 0 & 0 & b_{33} \end{bmatrix}} = \underset{I}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \tag{88}$$

5.3.8 Jacobi Diagonalization

หลักการตามวิธีของ Jocabi มีพื้นฐานที่กล่าวว่าค่าของ real symmetric matrix, $\tilde{A}$ ใดๆ จะมีเพียง real eigenvalues และ สามารถจะ diagonalized เข้าไปใน eigenvalue matrix $\Lambda = [\lambda_i]$ โดยวิธีการ iteration

ในวิธีของ Jacobi นี้ประกอบด้วยการ rotation matrices R หลายครั้ง ทำให้ off-diagonal elements ของ $\tilde{A}$ ถูกทำให้เป็นศูนย์ โดยการทำ iterations ซ้ำกันอีกจนกระทั่ง matrix $\tilde{A}$ ถูก diagonalized

วิธีการของ Jacobi ทำให้เกิดสมการมาตรฐานสำหรับ eigenproblem

$$(\tilde{A} - \lambda I)Y = 0 \tag{89}$$

ข้อดีที่สุดของวิธีการนี้ก็คือค่า eigenvalues และ eigenvectors จะหาได้พร้อมกันในเวลาเดียวกัน

ตัวอย่างของปัญหาสำหรับ second-order matrix นั่นคือ

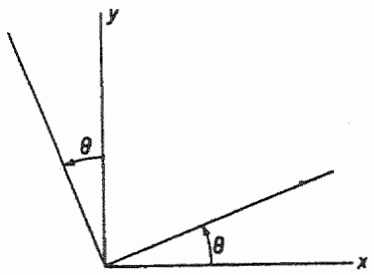
$$\tilde{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}$$

สำหรับค่า rotation matrix (R) ในกรณีนี้จะได้เป็น orthogonal matrix

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{90}$$

จากรูปที่ 14 ข้างล่างเป็นการทำการ transformation ของจุดพิกัด โดยหมุนแกนเป็นมุม θ ซึ่งหาความสัมพันธ์ของมุม θ ได้เป็น

$$\tan 2\theta = \frac{2a_{12}}{a_{11} - a_{22}} \tag{91}$$



รูปที่ 14 แสดงการ transformation ซึ่งหมุนแกนไปจากเดิมเป็นมุม θ

โปรแกรมคอมพิวเตอร์สำหรับ Jacobi diagonalization เทอม $\tan 2\theta = 2a_{ij} / (a_{ii} - a_{jj})$ จะถูกเปลี่ยนเป็น tanθ โดยได้ค่า identity

$$\tan 2\theta = \frac{2\tan\theta}{1-\tan^2\theta} = \frac{2a_{ij}}{a_{ii}-a_{jj}}$$

บทที่ 6
โปรแกรมแบบจำลองคลื่นการสั่นสะเทือน

โปรแกรมแบบจำลองที่นำเสนอในรายงานฉบับนี้มีจุดมุ่งหมายเพื่อช่วยการวิเคราะห์เชิง numerical ที่การคำนวณด้วยเครื่องคำนวณธรรมดาจะใช้เวลายาวนาน ในบางปัญหาจำเป็นอย่างยิ่งที่ต้องใช้คอมพิวเตอร์

6.1 โปรแกรมคลื่นการสั่นสะเทือน

โปรแกมที่อยู่ในแผ่นดิสก์หลังปกรายงานเป็นแบบจำลองคลื่นการสั่นสะเทือนมีโปรแกรมย่อยหลายโปรแกรม ซึ่งจัดเป็นเมนูอยู่ในโปรแกรมใหญ่ชื่อ Vib. Exe โปรแกรมเขียนด้วยภาษา C^{++} ซึ่งจะเขียนตาม Soluions ที่ได้อธิบายมาแล้วทั้งในระบบ SDF และ N-DOF บางส่วนของการคำนวณอาจจะยุ่งยากกว่าที่ได้แสดงไว้ในบทก่อน อย่างไรก็ตามคณะผู้วิจัยก็ได้ตรวจสอบกับเอกสารเกี่ยวกับการแก้ปัญหาของเรื่องการสั่นสะเทือน (ดูรายชื่อใน references) พบว่าผลลัพธ์อยู่ในเกณฑ์ที่ยอมรับได้

6.1.1 เมนูของโปรแกมการวิเคราะห์คลื่นการสั่นสะเทือน

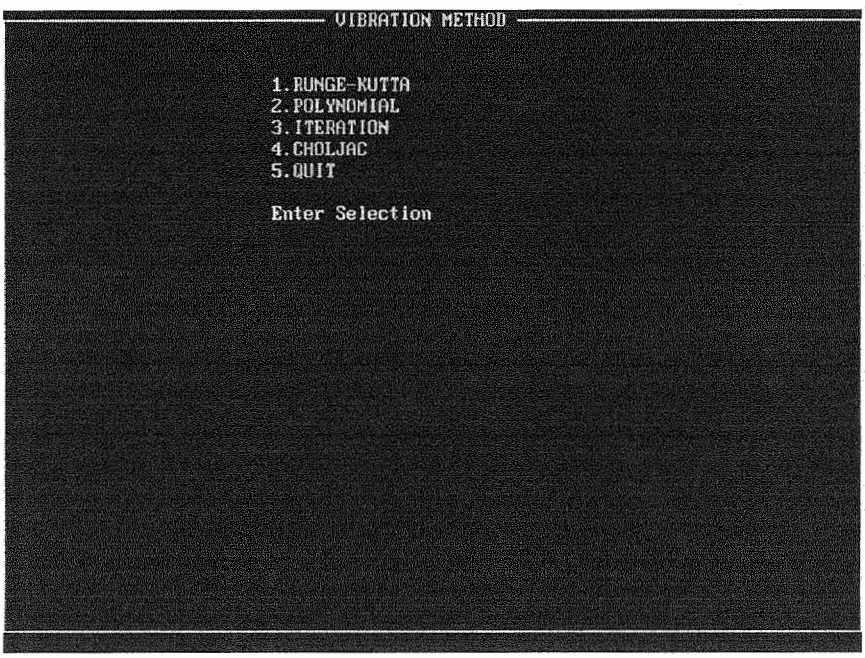
โปรแกรม Vib. Exe เป็นโปรแกรมใหญ่สำหรับวิเคราะห์คลื่นการสั่นสะเทือนหลายแบบ โดยผู้ใช้โปรแกรมสามารถเลือกใช้ได้ตามรูปแบบของเมนู ดังแสดงในรูปที่ 15

6.1.2 ชนิดของการวิเคราะห์

การวิเคราะห์แบ่งออกเป็น 2 ส่วน ได้แก่ การวิเคราะห์ระบบ Single Degree of Freedom กับการวิเคราะห์ระบบ Multidegree of Freedom

โดยแบ่งโปรแกรมการวิเคราะห์ออกเป็น 4 โปรแกรมย่อย คือ

1. โปรแกรม Kutta.Cpp
2. โปรแกรม Poly.Cpp
3. โปรแกรม Iterate.Cpp
4. โปรแกรม Choljac.Cpp



รูปที่ 15 วิธีการวิเคราะห์คลื่นการสั่นสะเทือนตามรูปแบบเมนู

โปรแกรมแรกใช้สำหรับการแก้ปัญหาของสมการคลื่นที่อยู่ในรูปแบบของ differential equation สำหรับระบบ SDF ส่วนโปรแกรมที่เหลือใช้การแก้ปัญหาของคลื่นเชิงพลศาสตร์ในรูปแบบของแมตริกซ์ สำหรับระบบ N-DOF

6.2 โปรแกรม Kutta

โปรแกรมนี้ช่วยแก้ปัญหา differential equation $m\ddot{x} + c\dot{x} + kx = f(t)$ สำหรับคลื่นการสั่นสะเทือนที่ปั่น Single Degree of Freedom และมีข้อมูลเข้าของพารามิเตอร์สำหรับ damping

และ excitation force ซึ่งจะสมมุติให้ m, c_1 และ k มีการแปรเปลี่ยนในลักษณะ linear ระหว่างจุดหลายจุดที่ปรากฏบน interval

ในการคำนวณและพิมพ์ผลแต่ละช่วงโปรแกรมสามารถรับค่าได้มากถึง 80 ค่า (จุด) ผลของการคำนวณจะได้ค่าการเปลี่ยนตำแหน่งของอนุภาค (particle diaplacement) และความเร็วของอนุภาค (particle velocity) ที่แปรเปลี่ยนกับเวลา (time) แล้วพลอตออกมาเป็นกราฟ โดยให้สามารถใช้ Microsoft Windows เพื่อความสะดวกในการ print กราฟด้วย laser printer

สภาวะเริ่มต้นของสมการกำหนดให้มีค่า $x(0) = \dot{x}(0) = 0$ โดยให้มีค่าคาบธรรมชาติเป็น

$$\tau_n = 2\pi\sqrt{\frac{m}{k}}$$

ส่วนค่าของช่วงเวลา (time interval, h) ที่เลือกไว้นั้นมีได้สูงสุด 80 ค่าของ t ผลของโปรแกรมนอกจากระบุค่าการเปลี่ยนตำแหน่งและความเร็วแล้วก็ยังพิมพ์ค่าของคาบธรรมชาติ (τ_n) และแอมพลิจูดสูงสุด (maximum amplitude, X_{max}) ไว้ให้ด้วย

6.3 โปรแกรม Poly

โปรแกรมนี้ช่วยแก้ปัญหาสำหรับคณิตศาสตร์ชั้นสูงที่จำเป็นสำหรับการวิเคราะห์หาผลลัพธ์ของคลื่นการสั่นสะเทือนในระบบ Mulitidegree of Freedom หัวข้อของการคำนวณในโปรแกรมมี 3 อย่าง จึงมี ทางเลือก (option) ของโปรแกรม 3 ทาง คือ

ก. ทางเลือกที่ 1 จากข้อมูลเข้าของค่า mass stiffness (M) กับค่า spring stiffness (K) ทำการคำนวณหาสัมประสิทธิ์ของสมการ polynomial จากค่าของ characteristic determinant $|M - \lambda k|$ โดยที่ผู้ใช้ป้อนค่า n × n matrixs โดยที่ matrix order มีค่าไม่เกิน 20 สำหรับค่าสัมประสิทธิ์ ที่หาได้ยังสามารถนำไปใช้ได้ในทางเลือกที่ 2

ข. ทางเลือกที่ 2 ต้องการหาค่ารากของสมการ polynomial (eigenvalue) :

$$c_{n+1}x^n + c_n x^{n-1} + \cdots + c_2 x + c_1 = 0$$

ผู้ใช้ทำการป้อนค่าข้อมูลเข้าที่รู้ค่าแล้ว(หรือคำนวณมาได้) ของค่า (n+1) coefficients, c_i จากนั้นโปรแกรมก็คำนวณหาค่าราก (eigenvalues) จากนั้นก็คำนวณต่อไปจนได้ค่า eigenvectors แล้วอาจส่งค่า eigenvectors นี้ไปยังทางเลือกที่ 3

ค. <u>ทางเลือกที่ 3</u> ต้องการหาค่า eigenvectors ของ M - λk ผู้ใช้ต้องป้อนข้อมูลเข้าของ M, k และ λ ซึ่งในการป้อนข้อมูลของค่า M และ K จะต้องมี matrix order ไม่เกิน 20 จากนั้นก็ป้อนค่า λ_i ของแต่ละ eigenvector (โดยอาจรับค่ามาจากทางเลือกที่ 2 ก็ได้) เมื่อข้อมูลเข้าครบ โปรแกรมก็จะคำนวณด้วยวิธี Gauss elimination เพื่อหาค่า eigenvector ของแต่ละค่า eigenvalue ที่ป้อนข้อมูลเข้าไป

6.4 <u>โปรแกรม Iterate</u>

โปรแกรมนี้ใช้คำนวณหา eigenvalues และ eigenvectors โดยการทำ matrix iteration ผู้ใช้โปรแกรมเป็นผู้ป้อนข้อมูลของ mass matrix, M และ stiffness matrix, K ตามจำนวนของ matrix order (ไม่เกิน 20)

วิธีการคำนวณเริ่มต้นที่สมการของการเคลื่อนที่ที่อยู่ในฟอร์มของ $K^{-1}MX = \lambda X$ และฟอร์มของ stiffness matrix $K = Q^TQ$ ถูก decomposed โดยใช้วิธีของ Choleski เพื่อหาค่า Q, Q^{-1} และ Q^{-T}

จากนั้นก็กำหนดค่า dynamic matrix $A = K^{-1}M = Q^{-1}Q^{-T}M$ ซึ่งค่า dynamic matrix ในกรณีนี้เป็น unsymmetric ในขณะเดียวกันค่า sweeping matrix, S ก็ถูกนำมาให้ กำหนดให้ S = I ซึ่งค่า I เป็น unit matrix สำหรับโหมดแรก

การหาโหมดต่อไปใช้วิธี matrix deflation ของ Gram-Schmidt orthogonalization เริ่มต้นโดยกระบวนการ iteration ของ deflated matrix $ASX_{i+1} = \bar{\lambda}_i X_i$ ซึ่งจะถูก normalized เพื่อลด order of matrix ไปทีละหนึ่ง จากนั้นนำค่าที่ได้มาทดสอบเชิง convergence เพื่อตัดสินใจว่าจะทำการ iteration ต่อไปหรือไม่ ?

เกณฑ์ของข้อกำหนดในการตัดสินใจก็ดูว่าความแตกต่างของ $|\bar{\lambda}_{i+1} - \bar{\lambda}_i|$ มีค่าน้อยกว่าค่า tolerance ถ้าไม่น้อยกว่าก็ทำ iteration ซ้ำ ถ้ามีค่าน้อยกว่าค่าของโหมดแรก, $\bar{\lambda}_1$ และค่า

eigenvector, $\varnothing$ ก็เสร็จสิ้นสมบูรณ์ โดยกระบวนการคำนวณจะต่อไปยังโหมดที่สอง (ถ้าต้องการ) เพื่อหาค่าของ sweeping matrix และทำ iteration ต่อไปจนได้ครบทุกโหมดตามที่ต้องการ

6.5 โปรแกรม Choljac

โปรแกรมนี้ใช้สำหรับ matrix decomposition กับ matrix diagonalization ในโปรแกรม Choljac มีการคำนวณ 3 รูปแบบ จึงกำหนดให้มีทางเลือกไว้ 3 ทาง คือ

ก. ทางเลือกที่ 1 เป็นโปรแกรมสำหรับหาผลคูณของ square matrix 2 ค่า ผู้ใช้เป็นผู้ป้อนค่า $n \times n$ matrics ของทั้ง mass matrix (M) และ stiffness matrix (K)

ข. ทางเลือกที่ 2 เป็นโปรแกรมสำหรับหา eigenvalues และ eigenvectors ของ $\tilde{A}$ - λI หรือ M - λI โดยที่ $\tilde{A}$ เป็น symmetry dynamic matrix ผู้ใช้เป็นผู้ป้อนค่า matrix $\tilde{A}$ จากนั้นโปรแกรมก็จะทำการคำนวณด้วยวิธีของ Jacobi ด้วยการทำ iteration เพื่อ diagonalize ค่า matrix $\tilde{A}$ จนได้ค่า eigenvalues, λ และค่า eigenvectors, $\varnothing$ เป็นผลลัพธ์ออกมา

ค. ทางเลือกที่ 3 เป็นโปรแกรมสำหรับหาค่า eigenvalues และ eigenvectors ของ M - λK โดยใช้วิธีการ 2 วิธีทั้งแบบ Choleski decomposition และแบบ Jacobi diagonalization

ผู้ใช้เป็นผู้ป้อนค่า M และ K จากนั้นโปรแกรมจะคำนวณโดยใช้วิธีของ Choleski เพื่อ decompose ค่า stiffness matrix $K = Q^T Q$ ให้เป็น dynamic matrix $A = Q^{-1} Q^T M$ จากนั้นก็ใช้วิธีของ Jacobi เพื่อ diagonalize ค่า matrix $\tilde{A}$ ทำให้ได้ค่า eigenvalues และค่า eigenvectors ของ $(M - \lambda K)\varnothing$ โดยที่ค่า eigenvalues ที่ได้มีความสัมพันธ์กับค่า natural frequencies, ω^2 ที่กำหนด

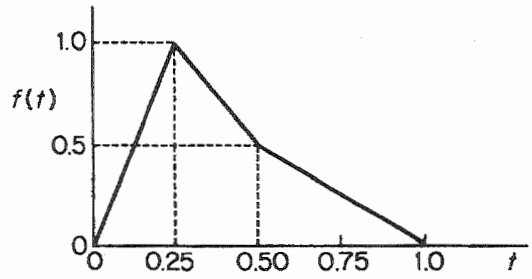
6.6 ตัวอย่างวิธีการใช้โปรแกรม

การคำนวณสำหรับคลื่นการสั่นสะเทือนได้มีการทดสอบข้อมูลมาจากเอกสารของ W.T. Thomson เรื่อง Theory of Vibration With Application : Fourth Edition จากการเปรียบเทียบผลลัพธ์ปรากฏว่าได้ผลตรงกันโปรแกรมแบบจำลอง Vib.Exe ที่เขียนขึ้นด้วยภาษา C^{++}

6.6.1 ตัวอย่างโปรแกรม Kutta

วิธีการคำนวณใช้วิธีของ Runge-Kutta โดยในตัวอย่างกำหนดให้ differential equation of motion เป็น $2\ddot{x} + 8\dot{x} + 100x = f(t)$

กำหนดให้มีสภาวะเริ่มต้นที่ $x(0) = \dot{x}(0) = 0$ โดยมีไดอะแกรมของแรง ดังรูปที่ 16



รูปที่ 16 ฟังก์ชันของแรงภายนอกเมื่อเทียบกับเวลา ในรูปมีความสัมพันธ์ของแรงกับเวลาอยู่ 4 จุด

ในตัวอย่างนี้ที่ปรากฏบนจอภาพของเครื่องคอมพิวเตอร์ ค่าคาบธรรมชาติ (τ_n) ได้ถูกกำหนดให้เท่ากับ $2\pi(m/k)^{0.5}$

รูปที่ 17 - 22 (หน้าที่ 59 - 61) แสดงถึงการ execute โปรแกรมที่ต้องป้อนข้อมูลเข้าและพลอตกราฟระหว่างค่าของตัวแปรที่พึ่งพิง (dependent variables) ได้แก่ค่า displacement และ velocity กับค่าของตัวแปรที่ไม่พึ่งพิง (independent variable) ได้แก่ time

6.6.2 ตัวอย่างโปรแกรม Poly

วิธีแก้ปัญหา Polynomial equation ของระบบ N-DOF นี้มี 3 ทางเลือก รูปที่ 23 แสดงชนิดของทางเลือกและเมนูของโปรแกรม Poly ส่วนรูปที่ 24 - 39 (หน้า 62 -70) แสดงวิธีการรับค่า การแสดงตัวเลขที่ถูกป้อนเป็นข้อมูลเข้า การแสดงผลลัพธ์และผลของการคำนวณในทางเลือกทั้ง 3 แบบ

ตัวอย่างที่แสดงของโปรแกรม Poly เป็นการหารากสมการมาตรฐานของ eigenvalue โดยกำหนดให้มีสมการของตัวอย่างเป็น

$$\bar{\lambda} \quad = \quad 1/\lambda \quad = \quad k/m\omega^2$$

ทำให้ได้ polynomial equation ที่แก้ปัญหาด้วยคอมพิวเตอร์ คือ

$$\bar{x}^3 - 5\bar{\lambda}^2 + 4.5\bar{\lambda} - 1 \quad = \quad 0$$

ซึ่งเป็นผลลัพธ์ในเชิงความเกี่ยวพันเป็น

$$\lambda^3 - 4.5\lambda^2 + 5\lambda - 1 \quad = \quad 0$$

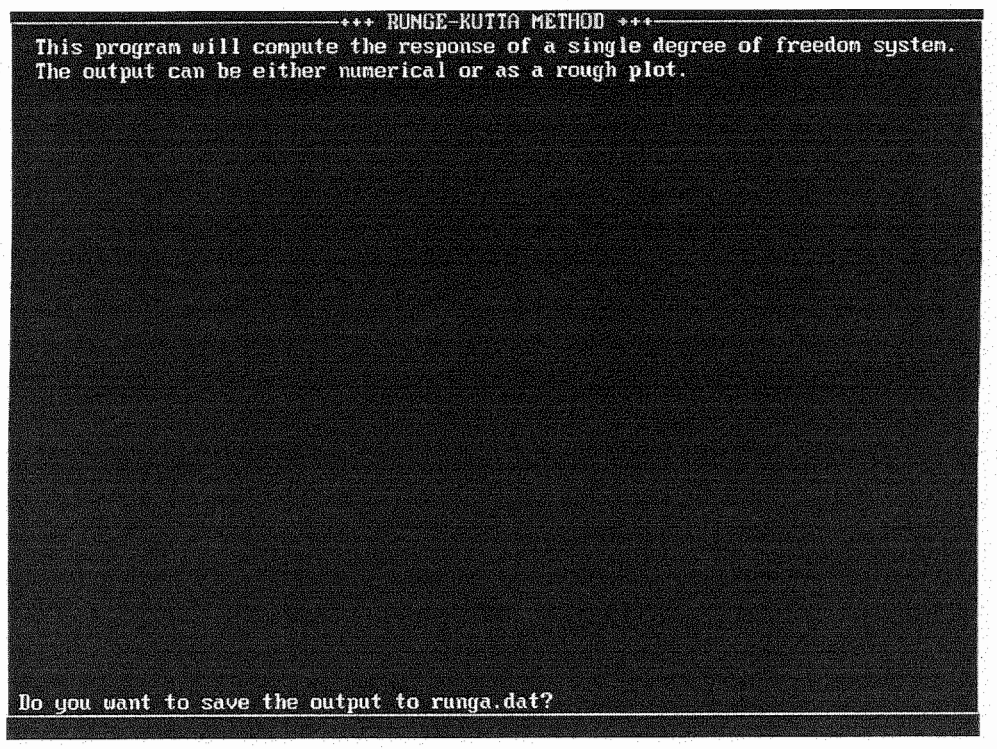
ตามที่กำหนดให้ค่า solution ของ $\lambda = m\omega^2/k$ จะได้ค่า eigenvalues และความถี่ธรรมชาติค่าต่างๆ เป็น

$$\lambda_1 = 0.25536 \qquad \omega_1 = 0.5.533\sqrt{k/m}$$

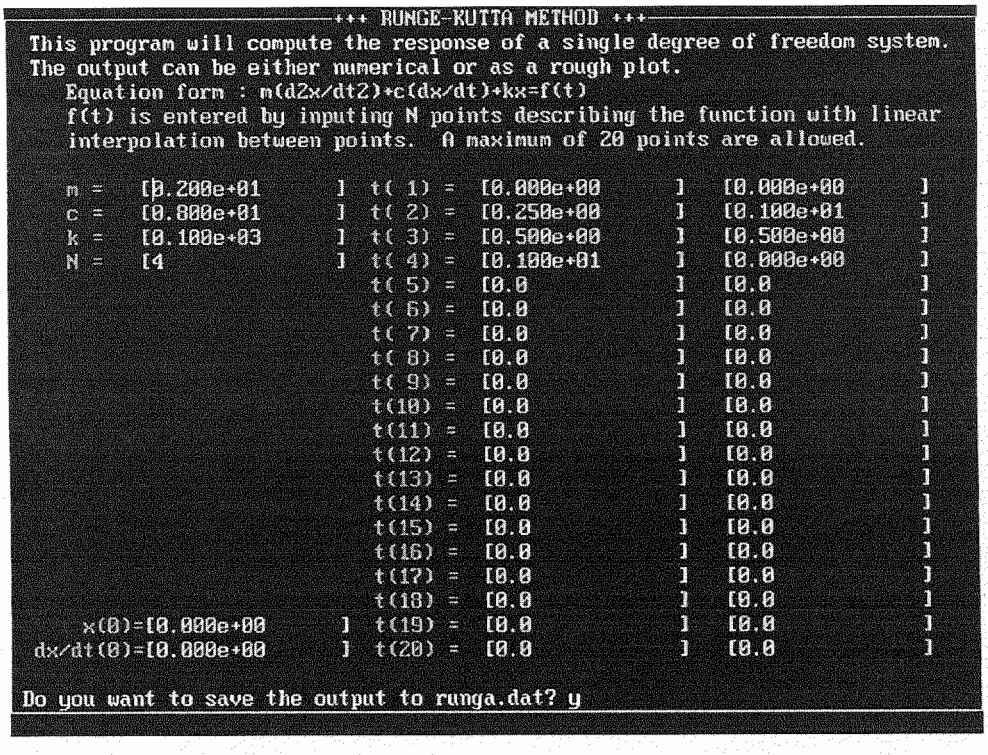
$$\lambda_2 = 1.3554 \qquad \omega_2 = 1.16422\sqrt{k/m}$$

$$\lambda_3 = 2.8892 \qquad \omega_3 = 1.69976\sqrt{k/m}$$

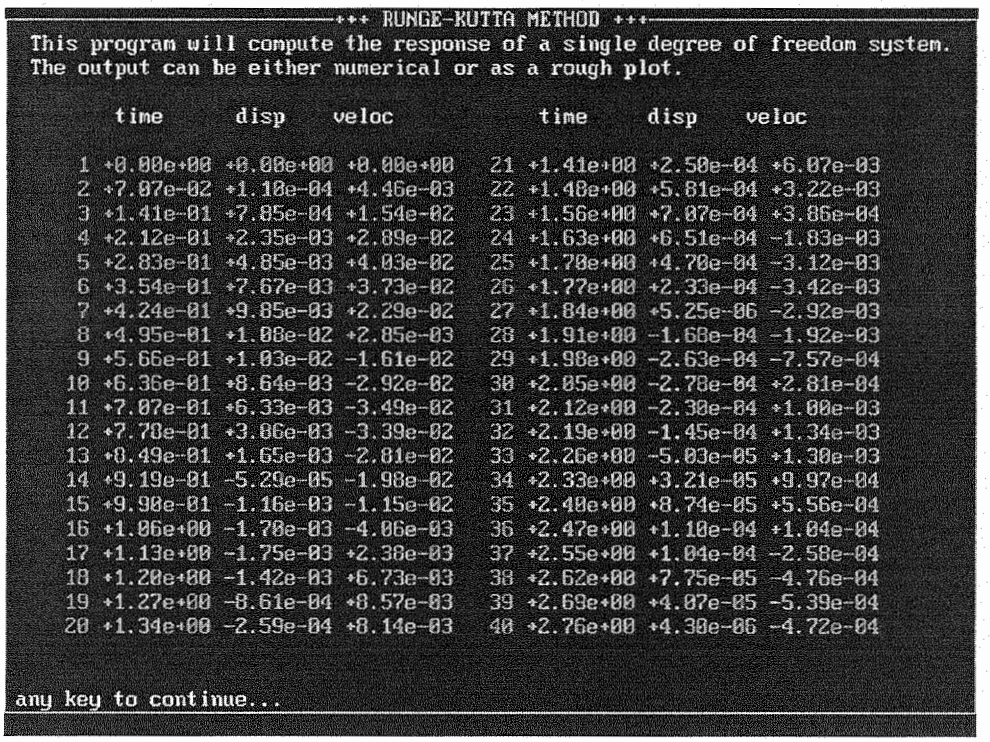
เมื่อทำการรวมค่าของ eigenvalues ทั้ง 3 ค่า จะได้ผลลัพธ์เป็น 4.50



รูปที่ 17 แสดงเมนูโปรแกรม Kutta สำหรับขั้นตอนและข้อจำกัด

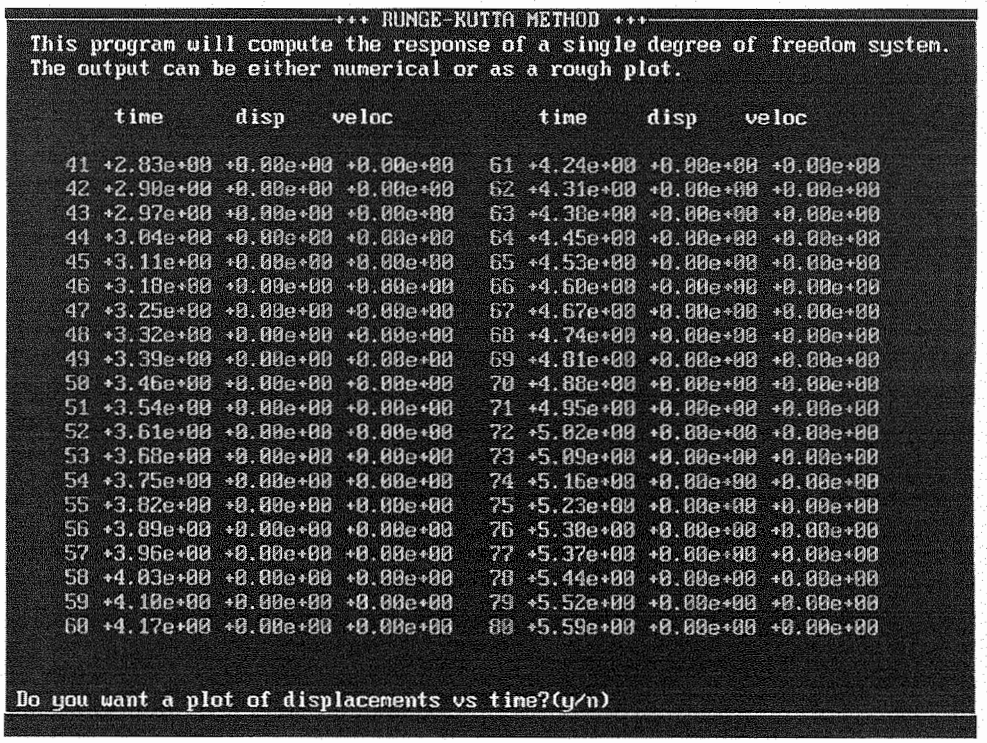


รูปที่ 18 แสดงการรับค่าข้อมูลเข้า ตามตัวอย่างโจทย์ ผู้ใช้เพียงเลือกระบบการเคลื่อนที่เชิงพลศาสตร์กับสภาวะเริ่มต้นที่จำเป็น



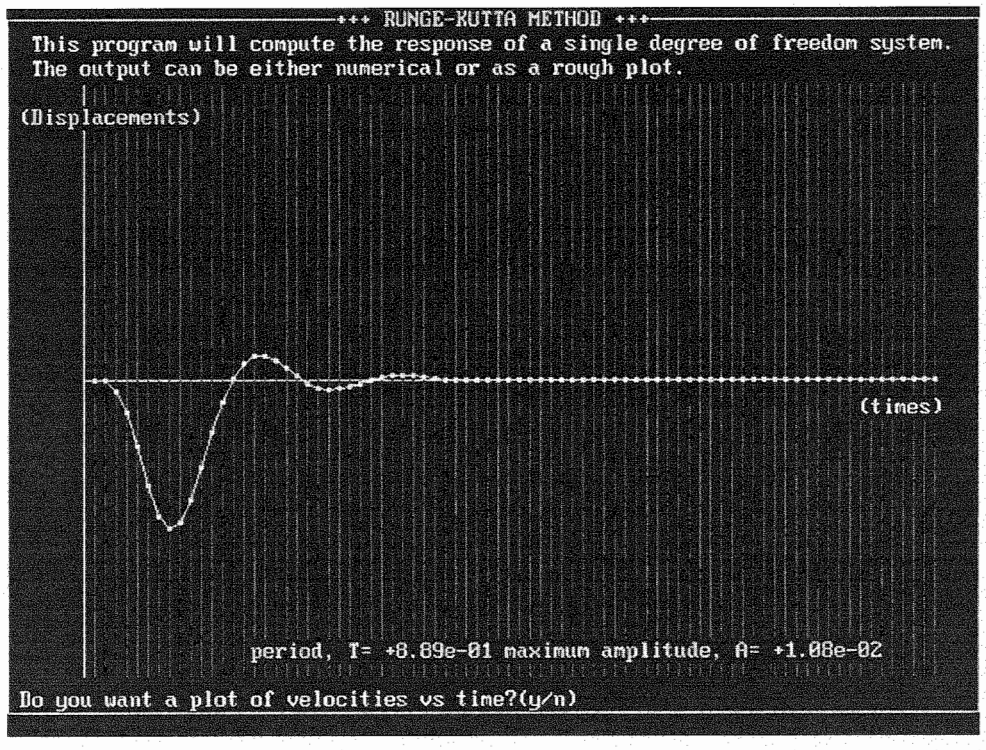
```
+++ RUNGE-KUTTA METHOD +++
This program will compute the response of a single degree of freedom system.
The output can be either numerical or as a rough plot.

      time      disp    veloc           time      disp    veloc

   1 +0.00e+00 +0.00e+00 +0.00e+00   21 +1.41e+00 +2.50e-04 +6.07e-03
   2 +7.07e-02 +1.10e-04 +4.46e-03   22 +1.48e+00 +5.81e-04 +3.22e-03
   3 +1.41e-01 +7.85e-04 +1.54e-02   23 +1.56e+00 +7.07e-04 +3.86e-04
   4 +2.12e-01 +2.35e-03 +2.89e-02   24 +1.63e+00 +6.51e-04 -1.83e-03
   5 +2.83e-01 +4.85e-03 +4.03e-02   25 +1.70e+00 +4.70e-04 -3.12e-03
   6 +3.54e-01 +7.67e-03 +3.73e-02   26 +1.77e+00 +2.33e-04 -3.42e-03
   7 +4.24e-01 +9.85e-03 +2.29e-02   27 +1.84e+00 +5.25e-06 -2.92e-03
   8 +4.95e-01 +1.08e-02 +2.85e-03   28 +1.91e+00 -1.68e-04 -1.92e-03
   9 +5.66e-01 +1.03e-02 -1.61e-02   29 +1.98e+00 -2.63e-04 -7.57e-04
  10 +6.36e-01 +8.64e-03 -2.92e-02   30 +2.05e+00 -2.78e-04 +2.81e-04
  11 +7.07e-01 +6.33e-03 -3.49e-02   31 +2.12e+00 -2.30e-04 +1.00e-03
  12 +7.78e-01 +3.86e-03 -3.39e-02   32 +2.19e+00 -1.45e-04 +1.34e-03
  13 +8.49e-01 +1.65e-03 -2.81e-02   33 +2.26e+00 -5.03e-05 +1.30e-03
  14 +9.19e-01 -5.29e-05 -1.98e-02   34 +2.33e+00 +3.21e-05 +9.97e-04
  15 +9.90e-01 -1.16e-03 -1.15e-02   35 +2.40e+00 +8.74e-05 +5.56e-04
  16 +1.06e+00 -1.70e-03 -4.06e-03   36 +2.47e+00 +1.10e-04 +1.04e-04
  17 +1.13e+00 -1.75e-03 +2.38e-03   37 +2.55e+00 +1.04e-04 -2.58e-04
  18 +1.20e+00 -1.42e-03 +6.73e-03   38 +2.62e+00 +7.75e-05 -4.76e-04
  19 +1.27e+00 -8.61e-04 +8.57e-03   39 +2.69e+00 +4.07e-05 -5.39e-04
  20 +1.34e+00 -2.59e-04 +8.14e-03   40 +2.76e+00 +4.30e-06 -4.72e-04


any key to continue...
```

รูปที่ 19 ผลลัพธ์ของการคำนวณที่ปรากฏบนจอภาพสำหรับค่า displacement กับค่า velocity ที่ได้จาก time increment 40 จุดแรก

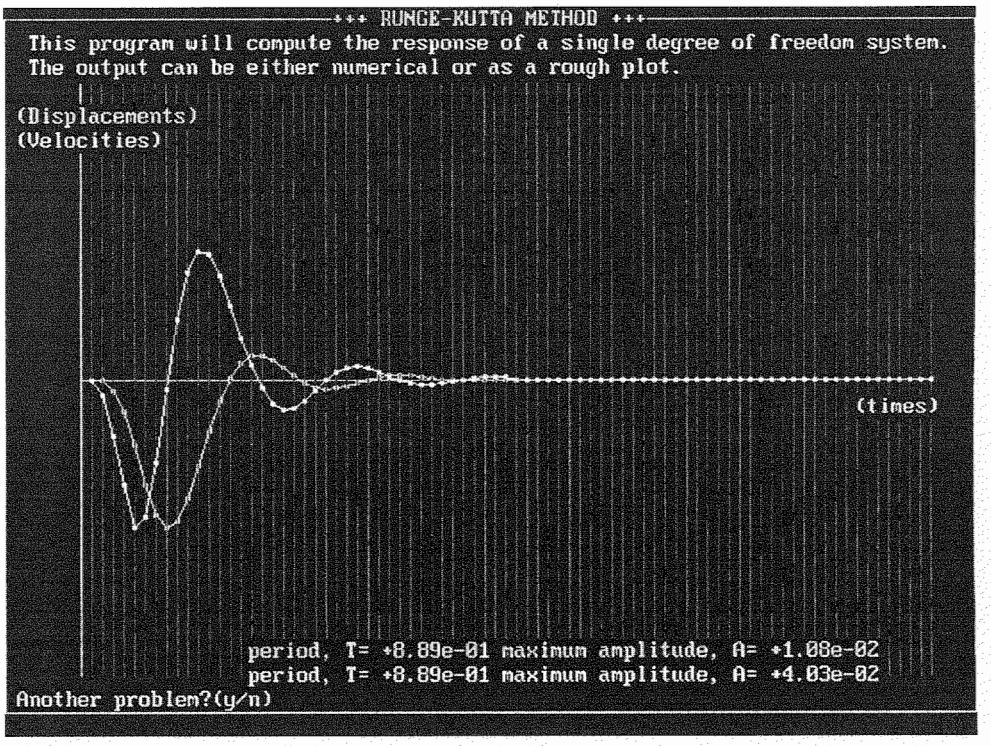


```
+++ RUNGE-KUTTA METHOD +++
This program will compute the response of a single degree of freedom system.
The output can be either numerical or as a rough plot.

      time      disp    veloc           time      disp    veloc

  41 +2.83e+00 +0.00e+00 +0.00e+00   61 +4.24e+00 +0.00e+00 +0.00e+00
  42 +2.90e+00 +0.00e+00 +0.00e+00   62 +4.31e+00 +0.00e+00 +0.00e+00
  43 +2.97e+00 +0.00e+00 +0.00e+00   63 +4.38e+00 +0.00e+00 +0.00e+00
  44 +3.04e+00 +0.00e+00 +0.00e+00   64 +4.45e+00 +0.00e+00 +0.00e+00
  45 +3.11e+00 +0.00e+00 +0.00e+00   65 +4.53e+00 +0.00e+00 +0.00e+00
  46 +3.18e+00 +0.00e+00 +0.00e+00   66 +4.60e+00 +0.00e+00 +0.00e+00
  47 +3.25e+00 +0.00e+00 +0.00e+00   67 +4.67e+00 +0.00e+00 +0.00e+00
  48 +3.32e+00 +0.00e+00 +0.00e+00   68 +4.74e+00 +0.00e+00 +0.00e+00
  49 +3.39e+00 +0.00e+00 +0.00e+00   69 +4.81e+00 +0.00e+00 +0.00e+00
  50 +3.46e+00 +0.00e+00 +0.00e+00   70 +4.88e+00 +0.00e+00 +0.00e+00
  51 +3.54e+00 +0.00e+00 +0.00e+00   71 +4.95e+00 +0.00e+00 +0.00e+00
  52 +3.61e+00 +0.00e+00 +0.00e+00   72 +5.02e+00 +0.00e+00 +0.00e+00
  53 +3.68e+00 +0.00e+00 +0.00e+00   73 +5.09e+00 +0.00e+00 +0.00e+00
  54 +3.75e+00 +0.00e+00 +0.00e+00   74 +5.16e+00 +0.00e+00 +0.00e+00
  55 +3.82e+00 +0.00e+00 +0.00e+00   75 +5.23e+00 +0.00e+00 +0.00e+00
  56 +3.89e+00 +0.00e+00 +0.00e+00   76 +5.30e+00 +0.00e+00 +0.00e+00
  57 +3.96e+00 +0.00e+00 +0.00e+00   77 +5.37e+00 +0.00e+00 +0.00e+00
  58 +4.03e+00 +0.00e+00 +0.00e+00   78 +5.44e+00 +0.00e+00 +0.00e+00
  59 +4.10e+00 +0.00e+00 +0.00e+00   79 +5.52e+00 +0.00e+00 +0.00e+00
  60 +4.17e+00 +0.00e+00 +0.00e+00   80 +5.59e+00 +0.00e+00 +0.00e+00


Do you want a plot of displacements vs time?(y/n)
```

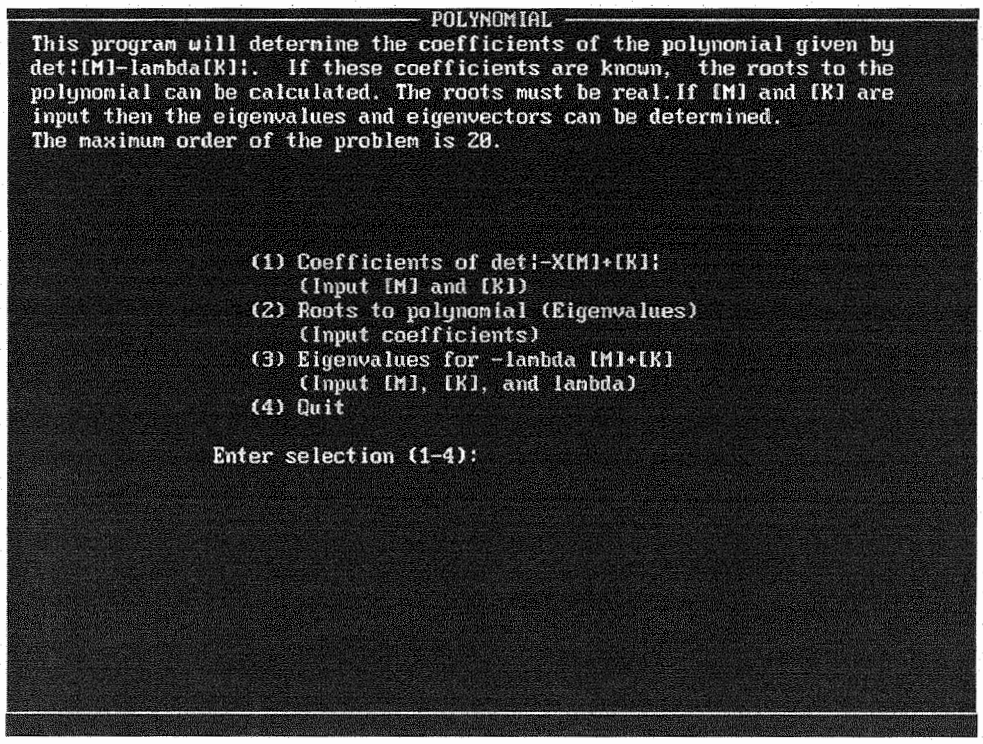
รูปที่ 20 ผลลัพธ์ของการคำนวณที่ปรากฏบนจอภาพสำหรับค่า displacement กับค่า velocity ที่ได้จาก time increment 40 จุดหลัง



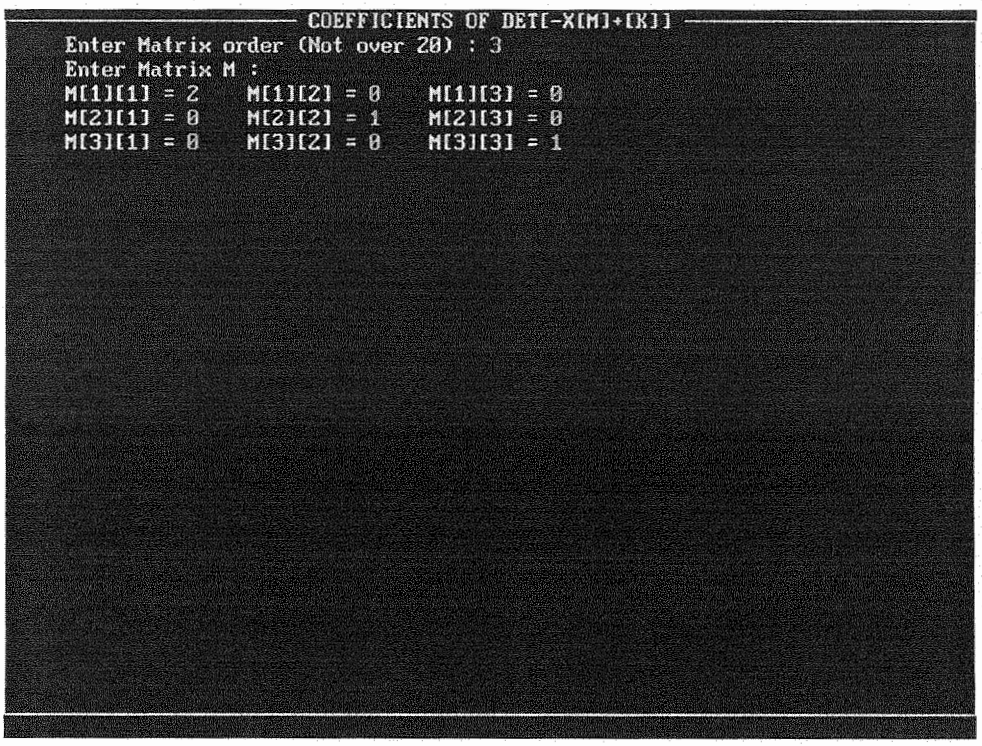
รูปที่ 21 เป็น response spectrum ที่แสดงความสัมพันธ์ของ displacement กับ time



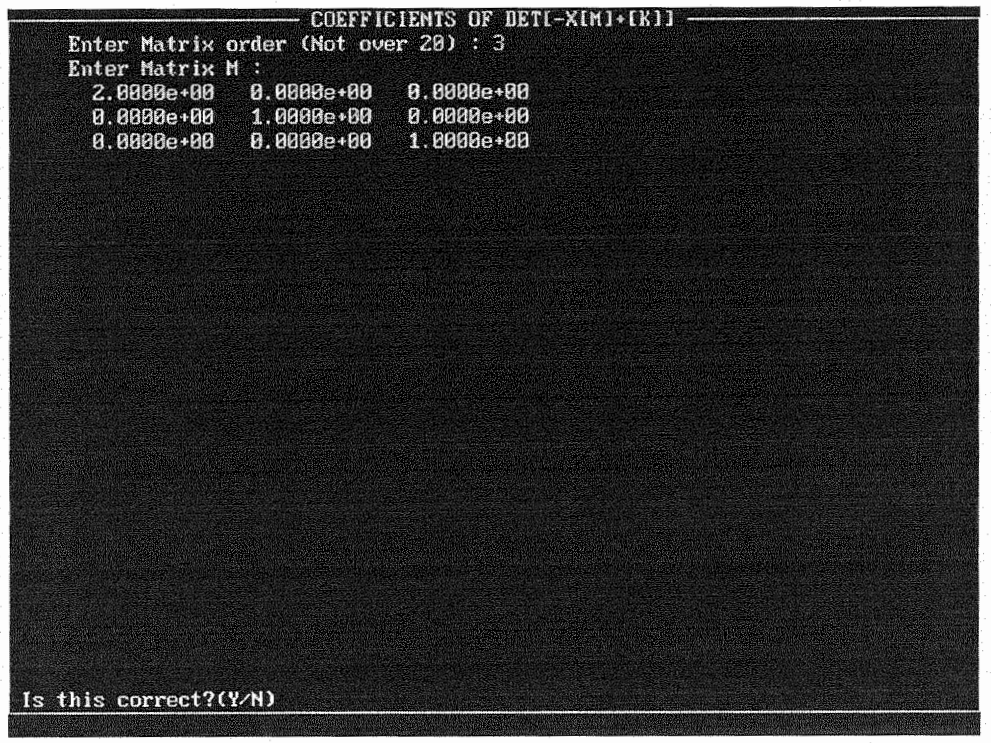
รูปที่ 22 เป็น response spectrum ที่ที่เปรียบเทียบระหว่าง quantities ของ displacement กับ time และระหว่าง quantities ของ velocity กับ time



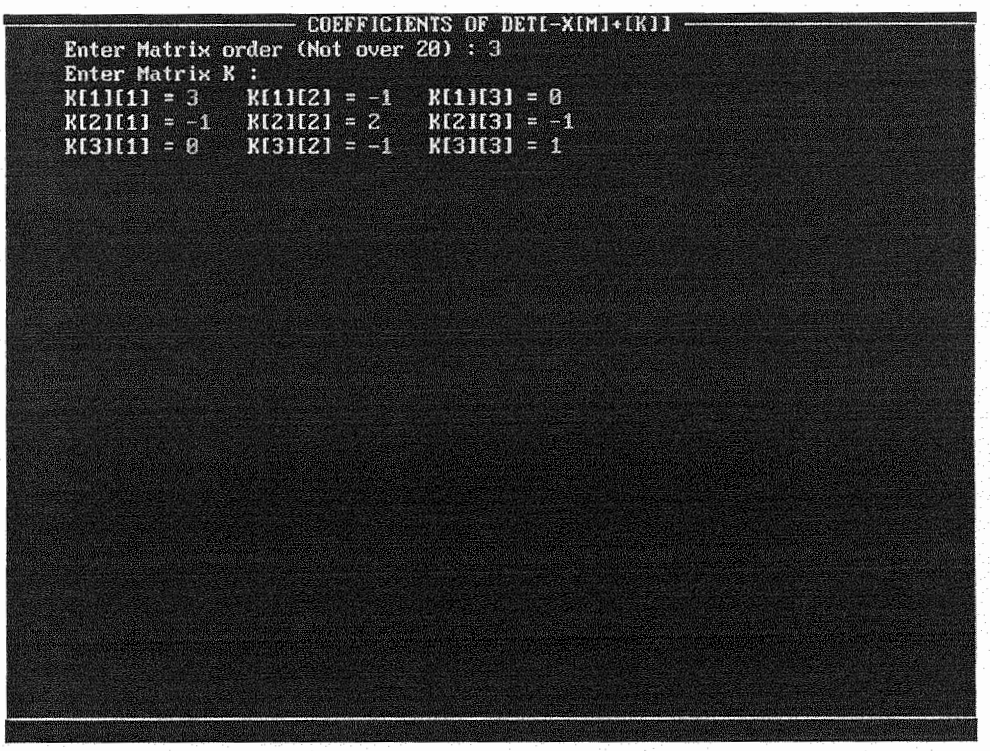
รูปที่ 23 เมนูของโปรแกรม Poly แสดงขั้นตอนการคำนวณและข้อจำกัด



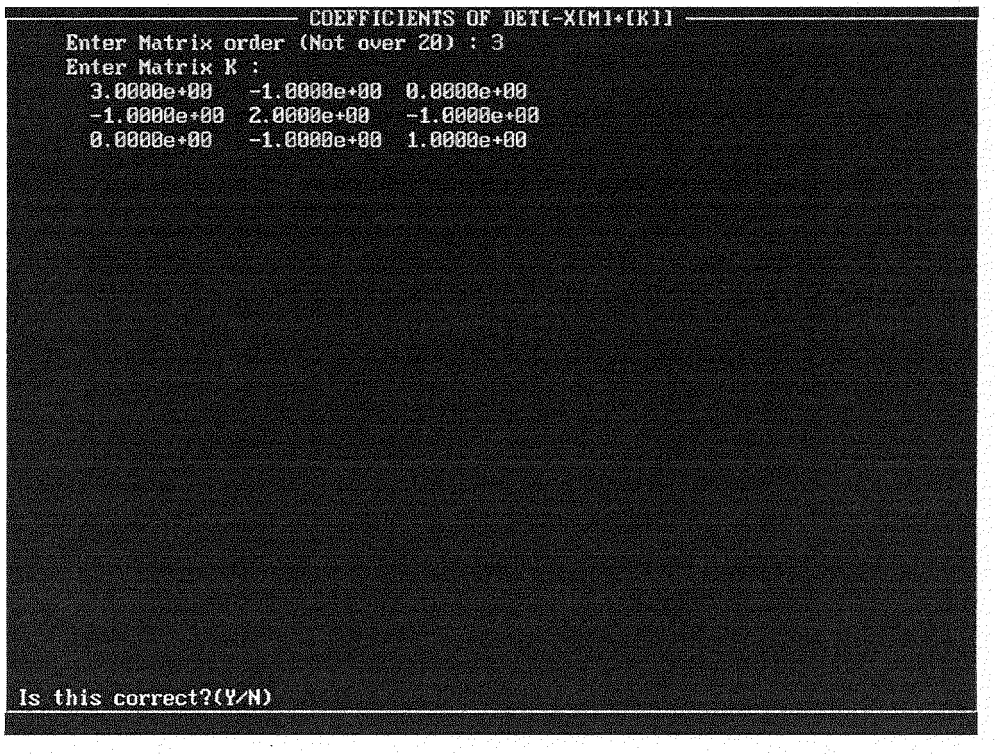
รูปที่ 24 โปรแกรม Poly ทางเลือกที่ 1 แสดงการรับค่า input matrix [M]
เพื่อหาค่า coefficients



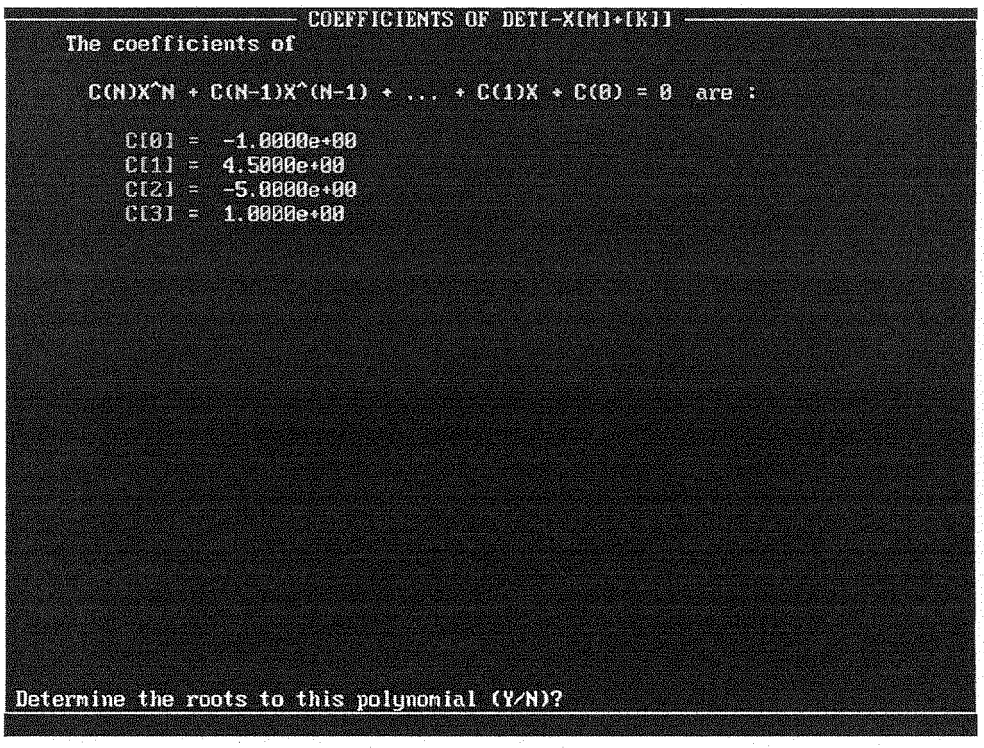
รูปที่ 25 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของ input matrix [M]
สำหรับโปรแกรม Poly ทางเลือกที่ 1



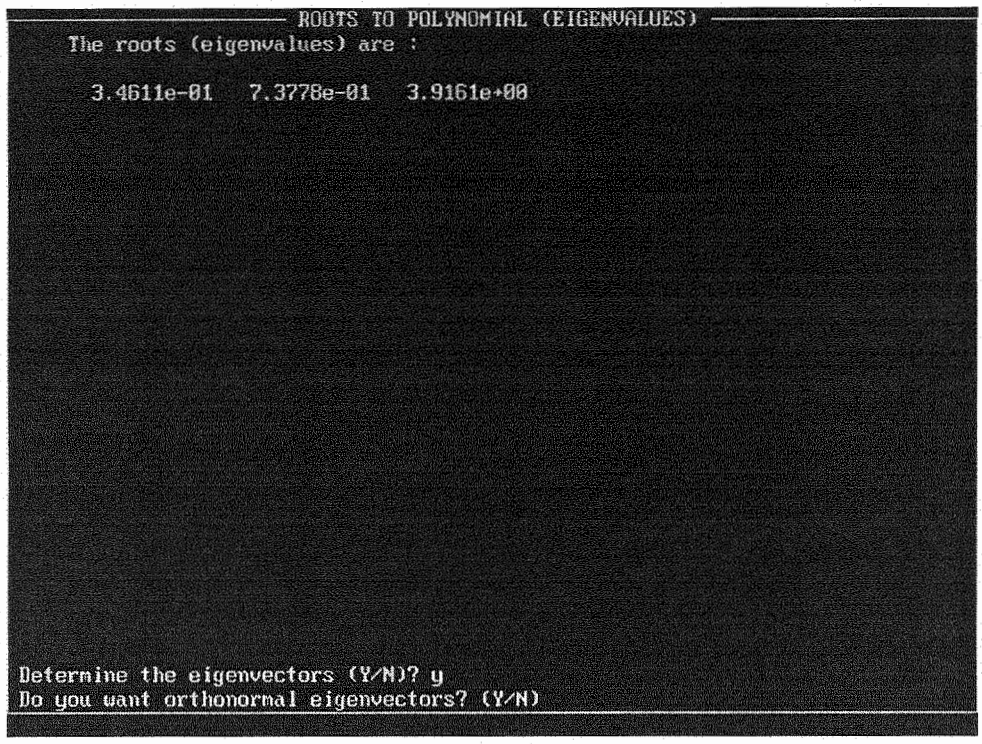
รูปที่ 26 โปรแกรม Poly ทางเลือกที่ 1 แสดงการรับค่า input matrix [K]
เพื่อหาค่า coefficients



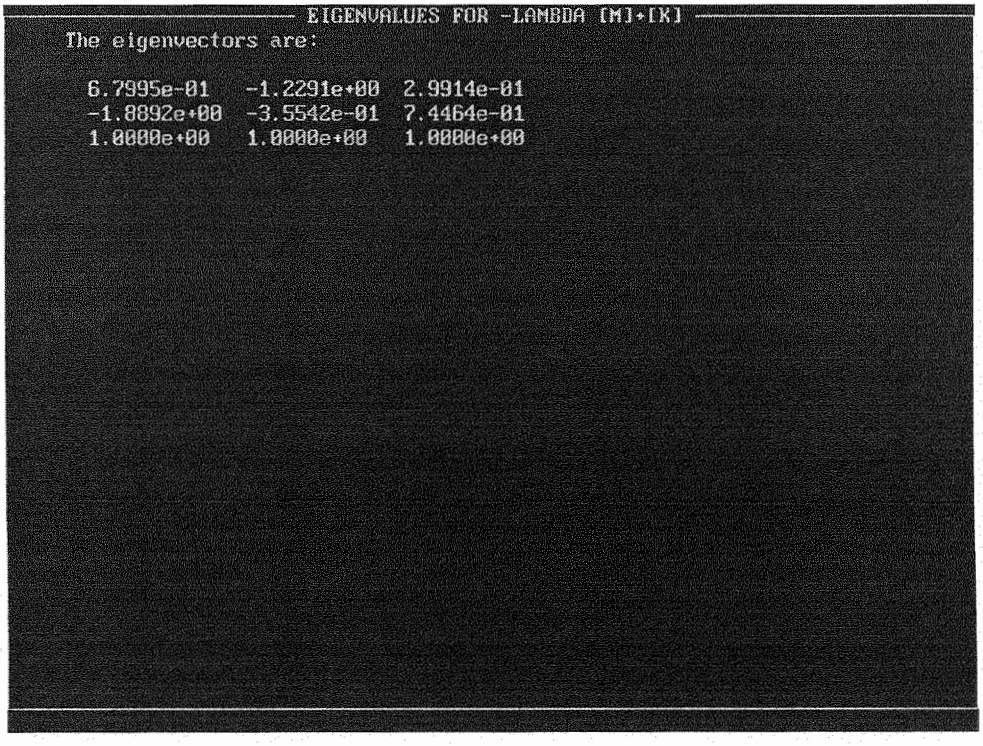
รูปที่ 27 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของ input matrix [K]
สำหรับโปรแกรม Poly ทางเลือกที่ 1



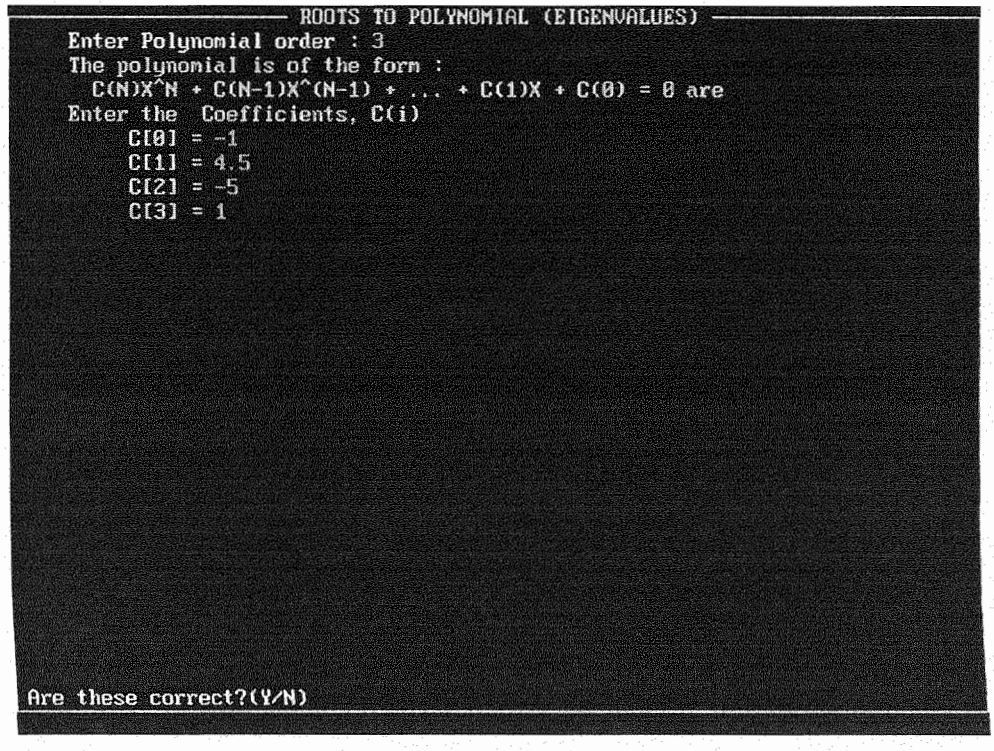
รูปที่ 28 แสดงผลลัพธ์ค่า coefficients ที่คำนวณได้จาก determinant [-x[M] + [K]]



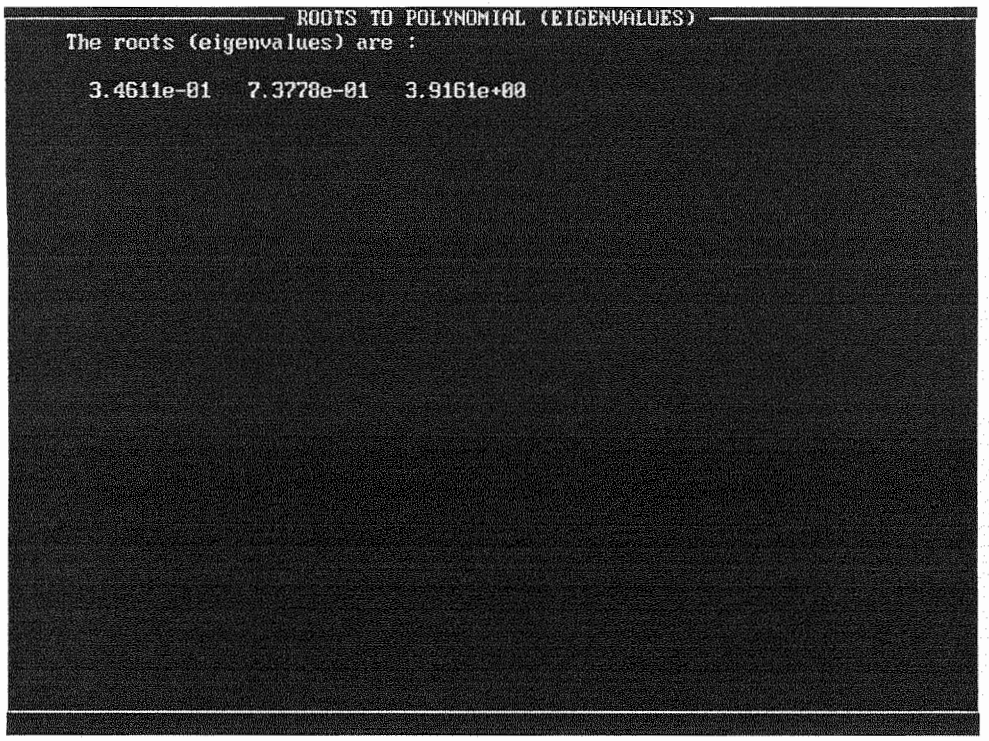
รูปที่ 29 แสดงผลลัพธ์ค่า eigenvalues ที่คำนวณได้จาก input ของ matrix [M] กับ matrix [K] จากโปรแกรม Poly ทางเลือกที่ 2



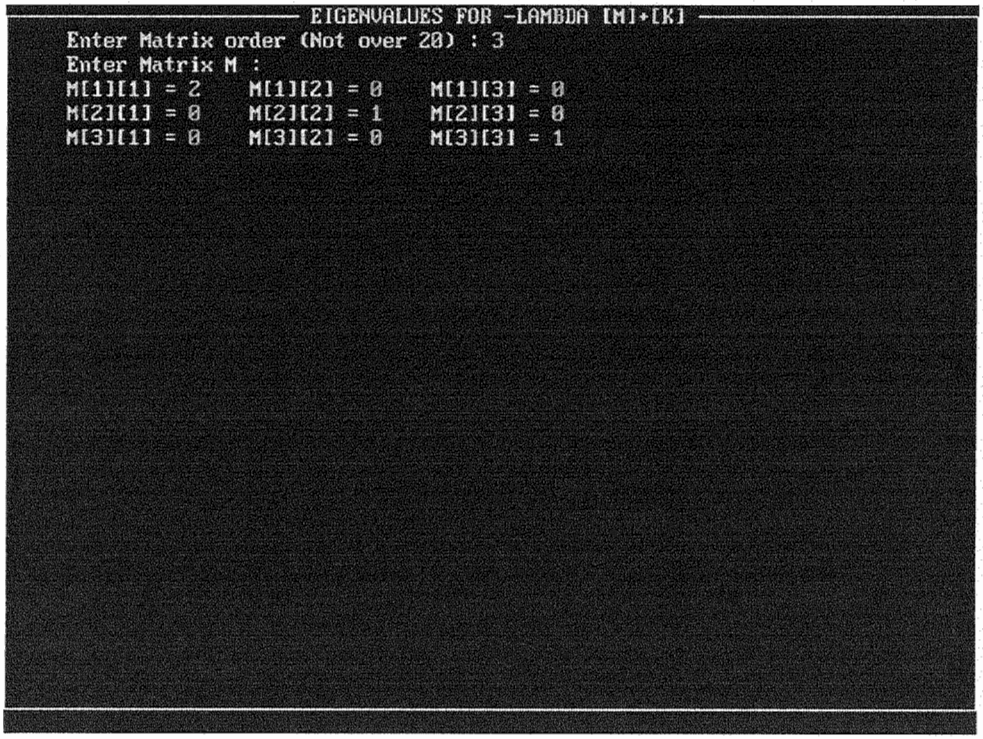
รูปที่ 30 แสดงผลลัพธ์ค่า eigenvectors ที่คำนวณได้จาก input ของ matrix [M] กับ matrix [K] จากโปรแกรม Poly ทางเลือกที่ 2



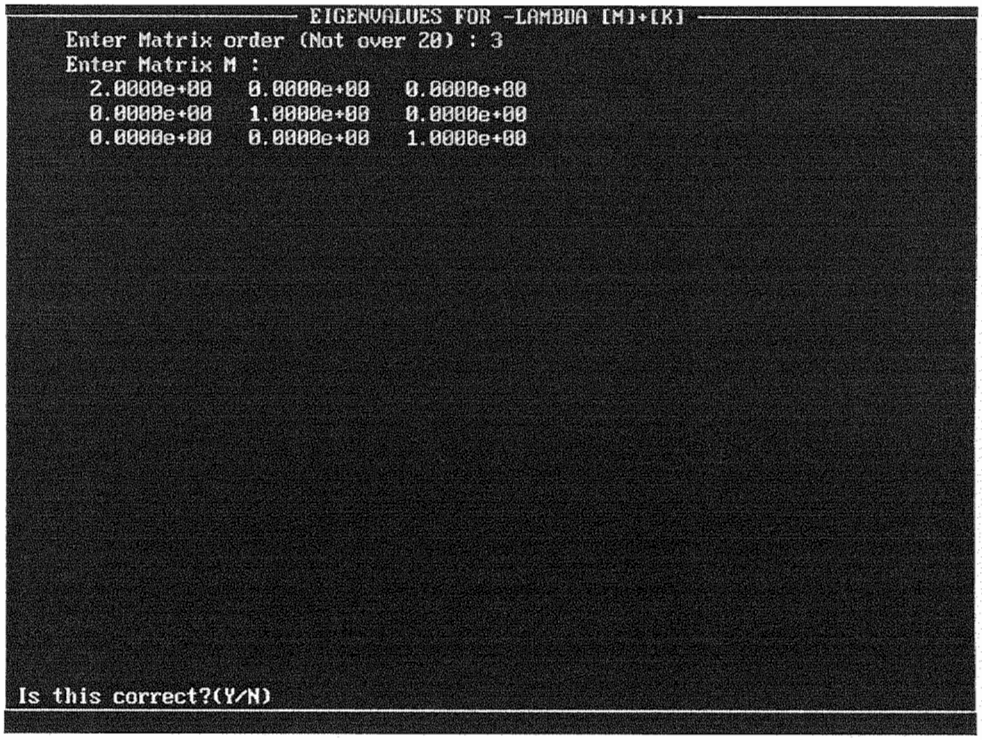
รูปที่ 31 แสดงการรับค่า coefficients เพื่อหาค่า eigenvalues จากโปรแกรม Poly ทางเลือกที่ 2



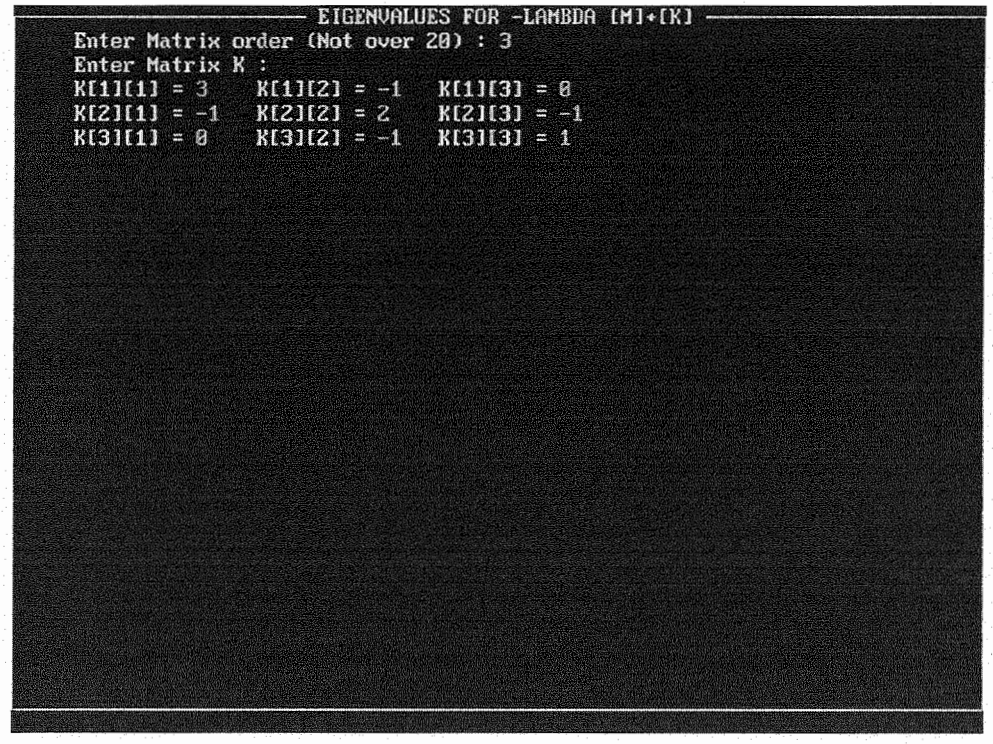
รูปที่ 32 แสดงผลลัพธ์ eigenvalues ที่คำนวณได้จากค่า input ของ coefficients จากโปรแกรม Poly ทางเลือกที่ 2



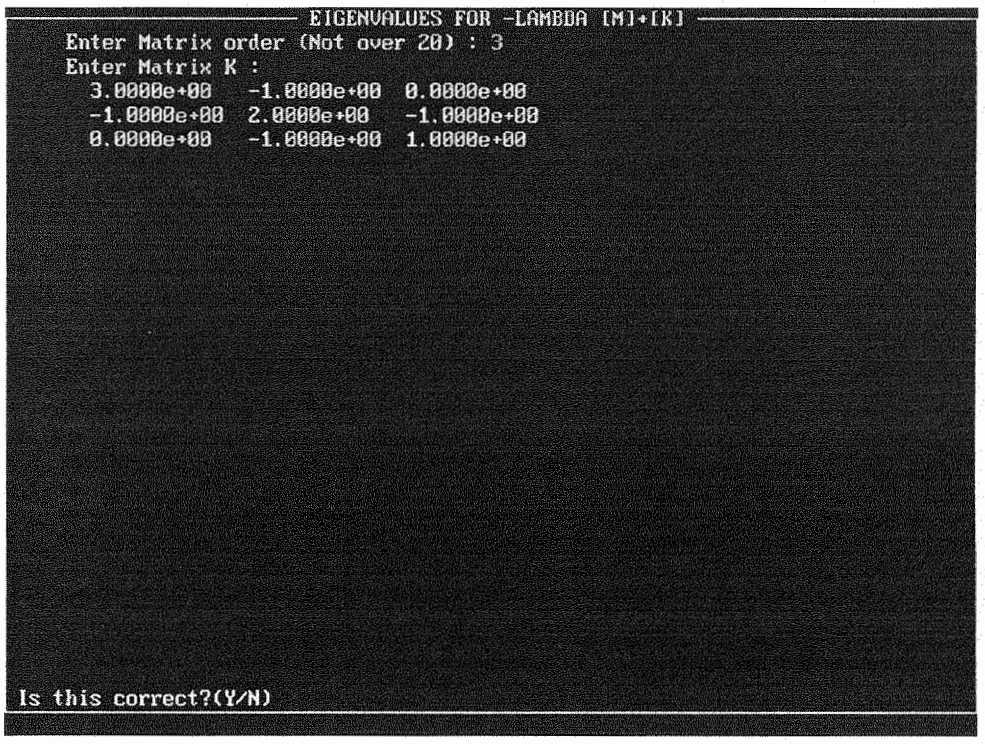
รูปที่ 33 แสดงการรับค่า input matrix [M] เพื่อใช้ในการคำนวณ
สำหรับโปรแกรม Poly ทางเลือกที่ 3



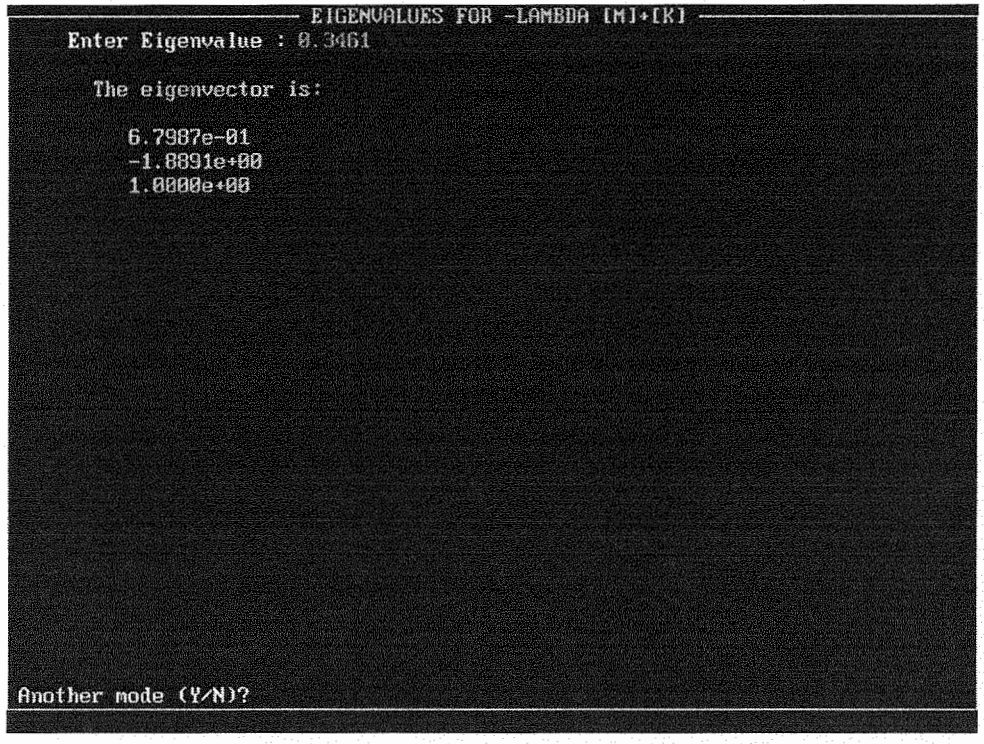
รูปที่ 34 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของค่า input matrix [M]
สำหรับโปรแกรม Poly ทางเลือกที่ 3



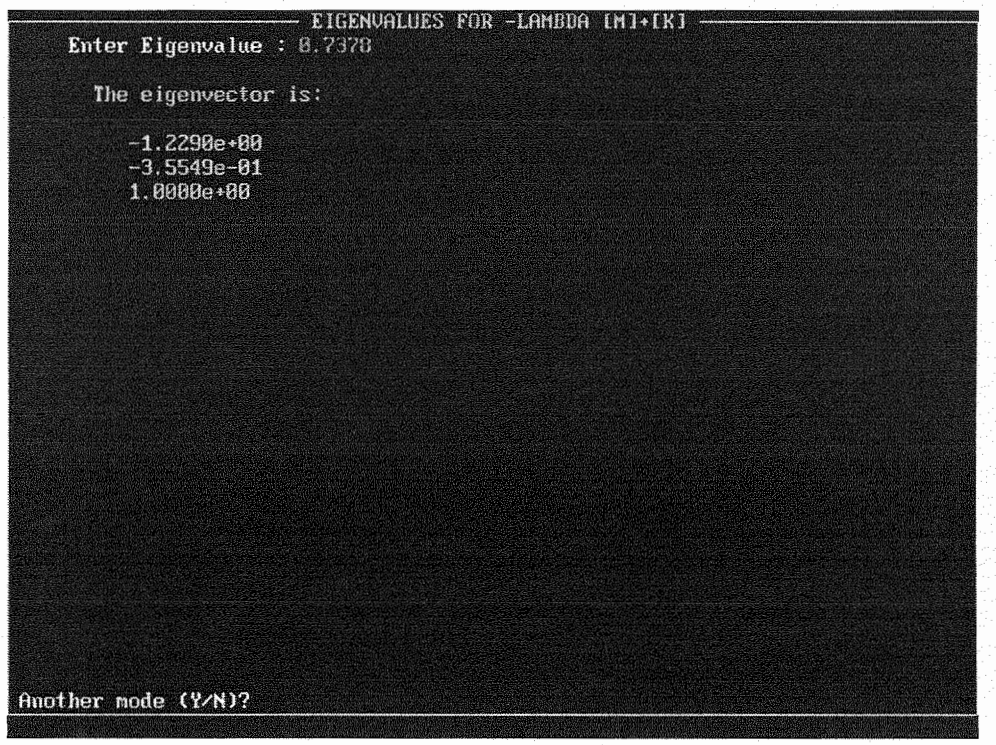
รูปที่ 35 แสดงการรับค่าของ input matrix [K] เพื่อใช้ในการคำนวณหาค่า eigenvalues และ eigenvectors สำหรับโปรแกรม Poly ทางเลือกที่ 3



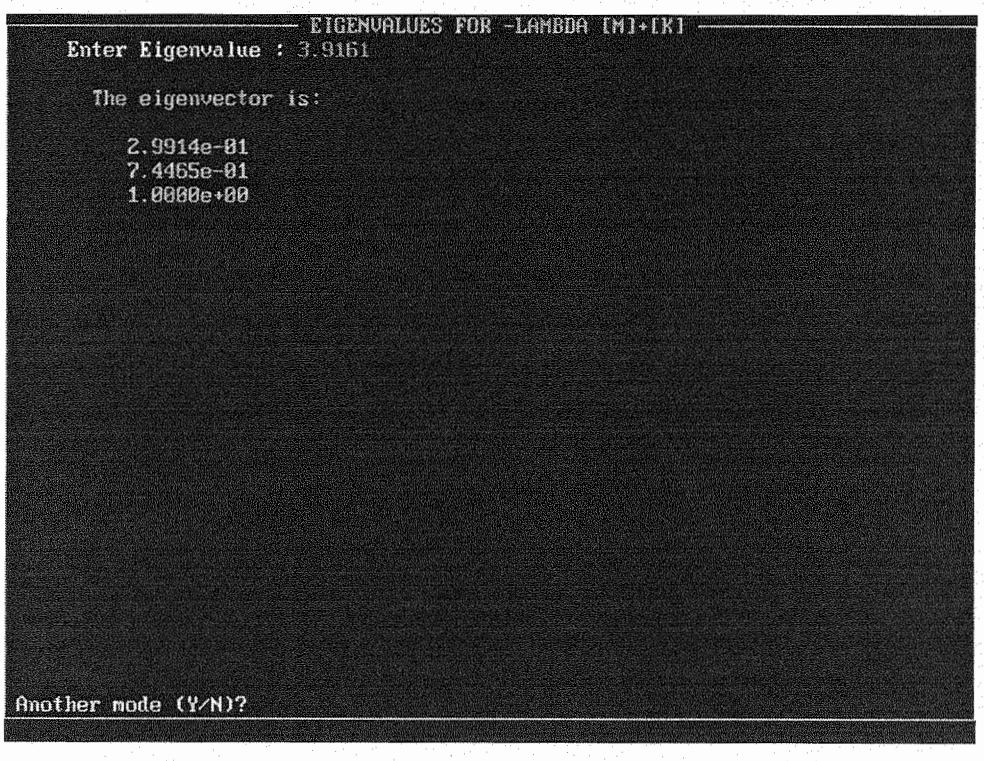
รูปที่ 36 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของค่า input matrix [K] สำหรับโปรแกรม Poly ทางเลือกที่ 3



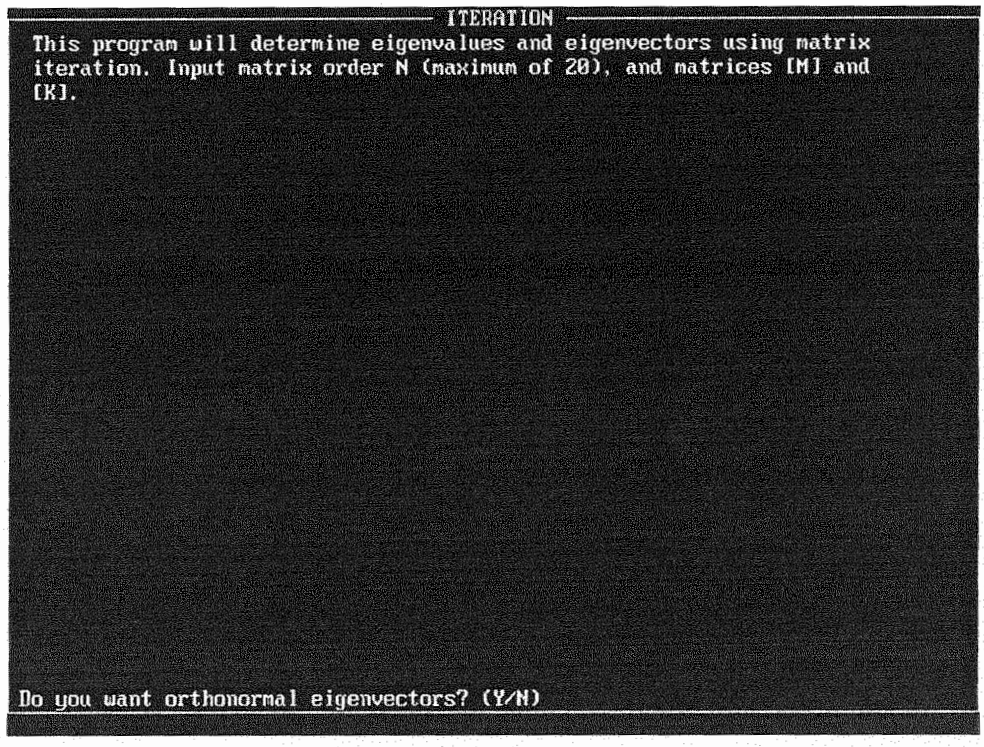
รูปที่ 37 แสดงผลลัพธ์ค่า eigenvector เมื่อกำหนดให้ค่า eigenvalue เป็น 0.3461 โปรแกรม Poly ทางเลือกที่ 3



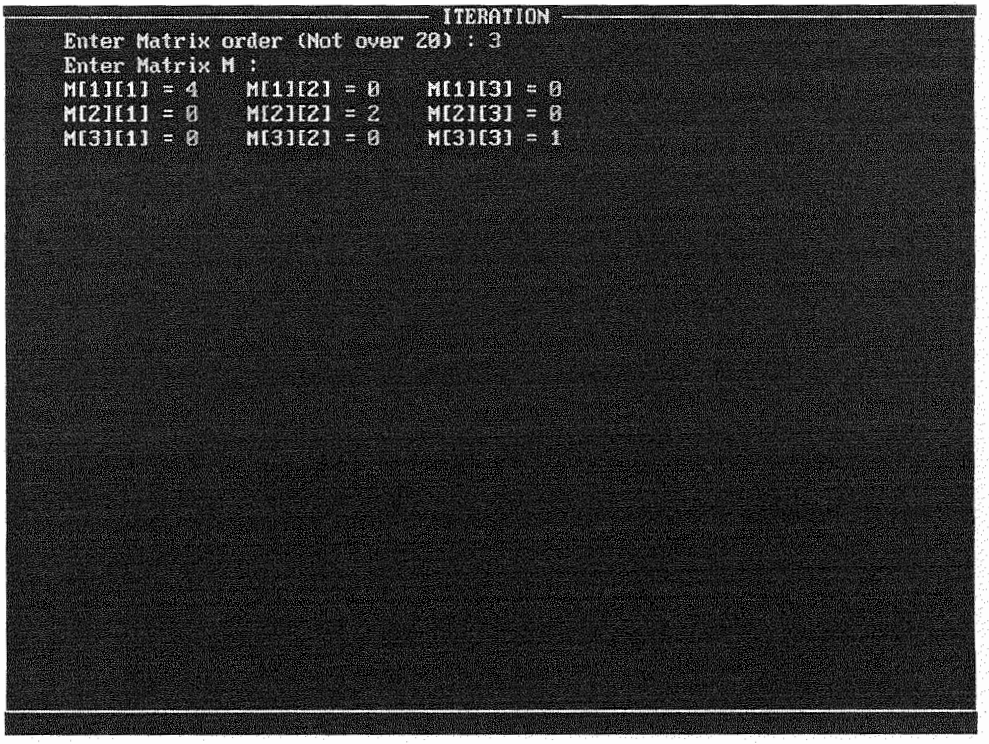
รูปที่ 38 แสดงผลลัพธ์ค่า eigenvector เมื่อกำหนดให้ค่า eigenvalue เป็น 0.7378 ของโปรแกรม Poly ทางเลือกที่ 3



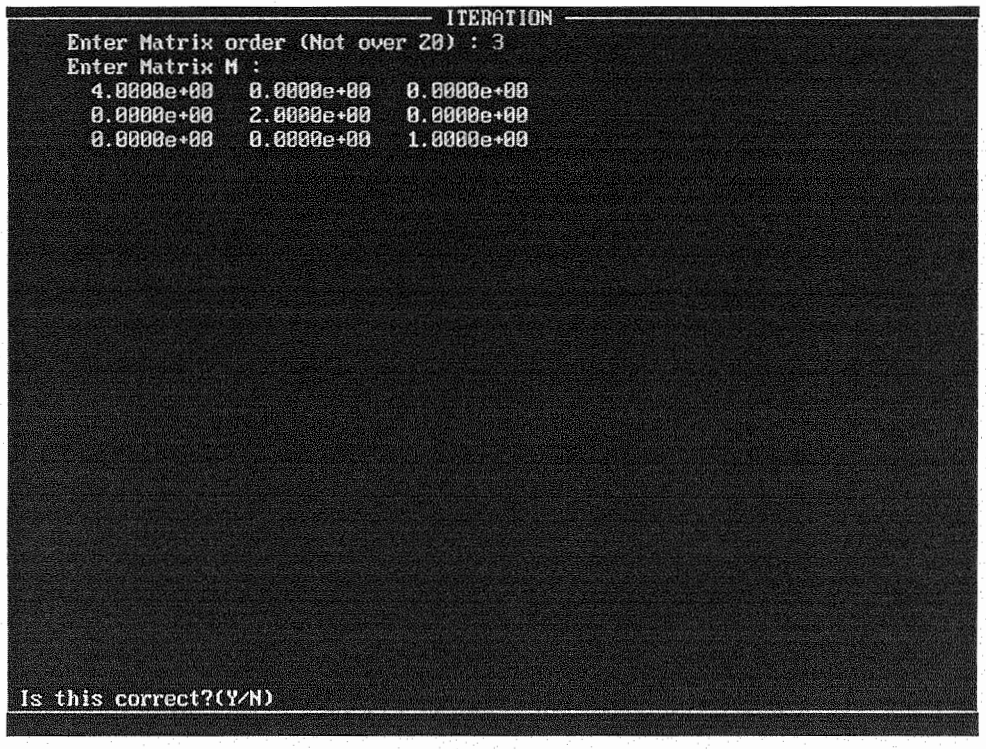
รูปที่ 39 แสดงผลลัพธ์ค่า eigenvector เมื่อกำหนดให้ค่า eigenvalue เป็น 3.9161 ของโปรแกรม Poly ทางเลือกที่ 3



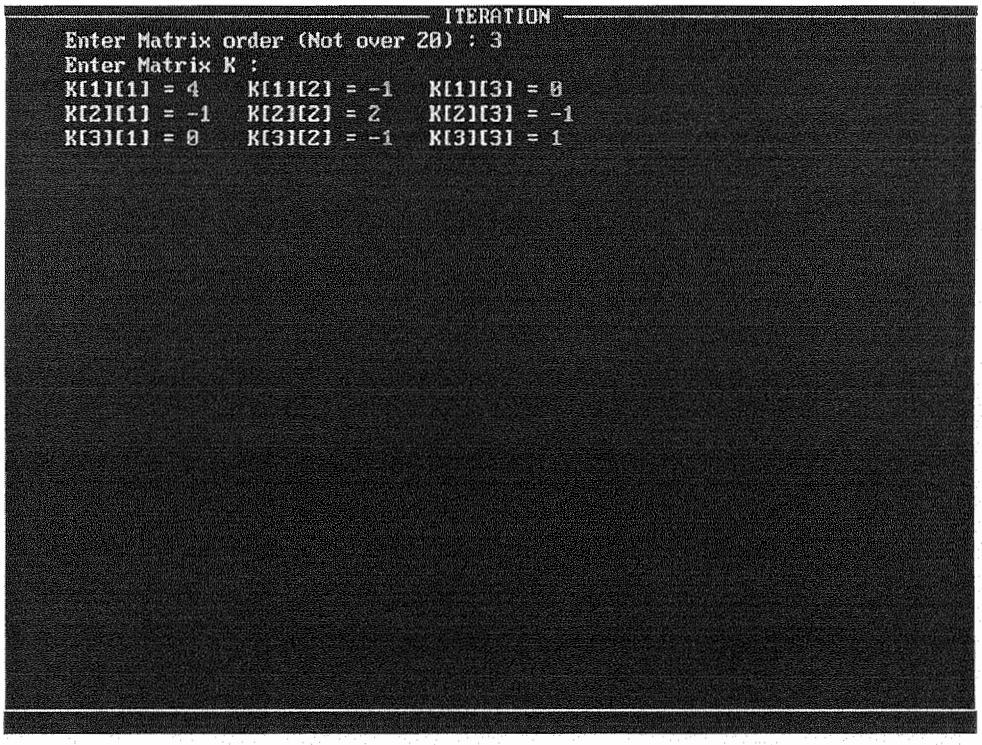
รูปที่ 40 เมนูของโปรแกรม Iteration แสดงขั้นตอนการคำนวณและข้อจำกัด



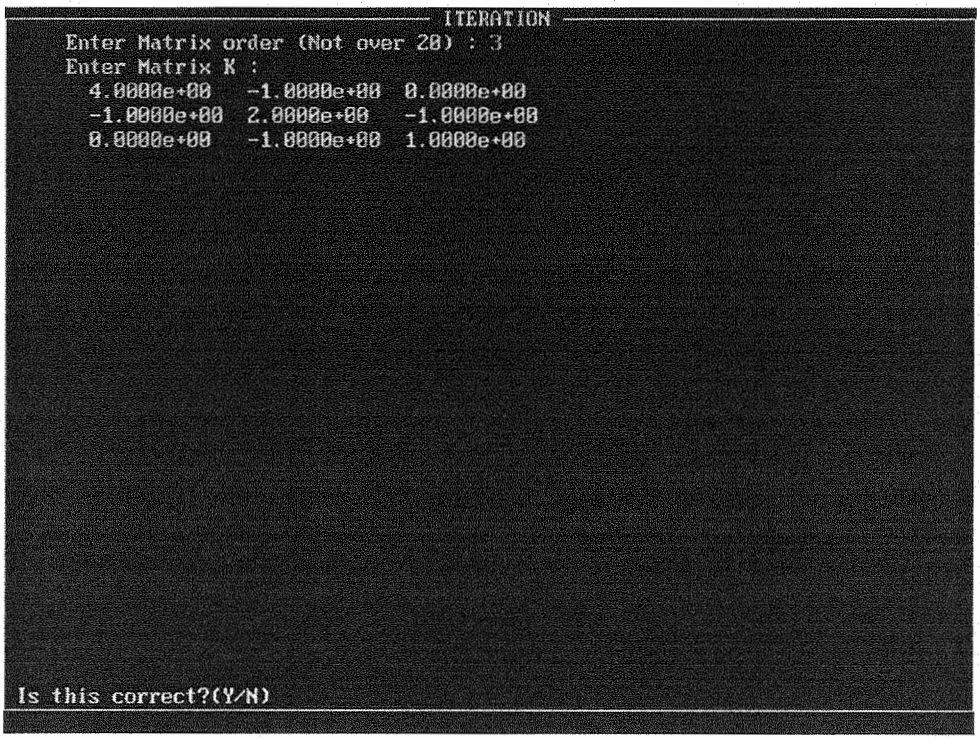
รูปที่ 41 แสดงการรับค่าของ input matrix [M] เพื่อใช้ในการคำนวณ สำหรับโปรแกรม Iteration



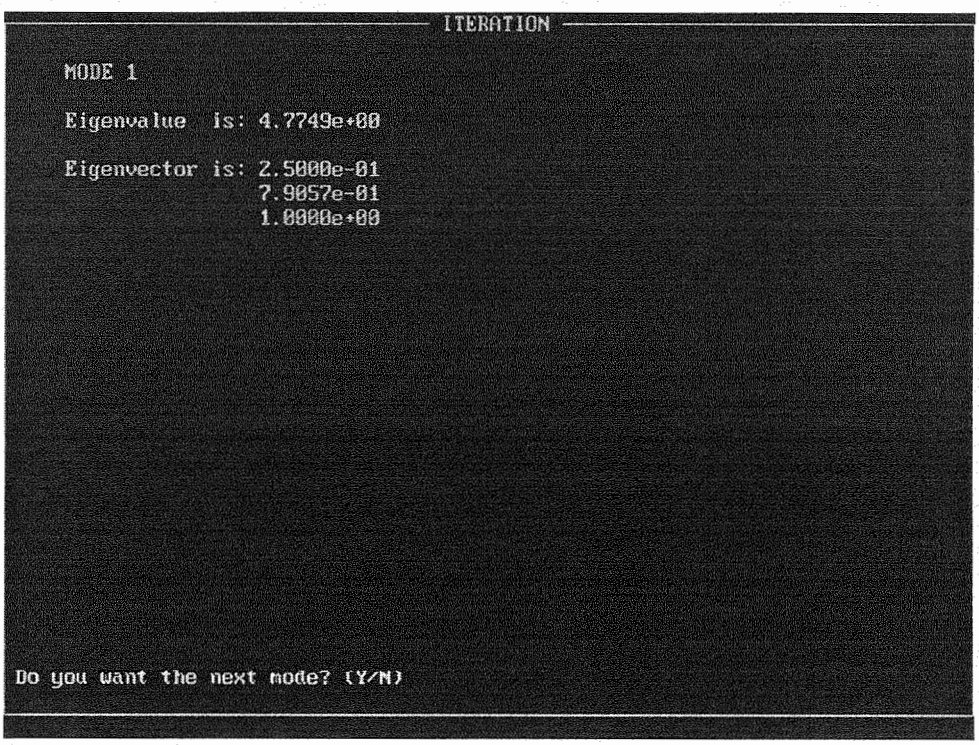
รูปที่ 42 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของค่า
input matrix [K] สำหรับโปรแกรม Iteration



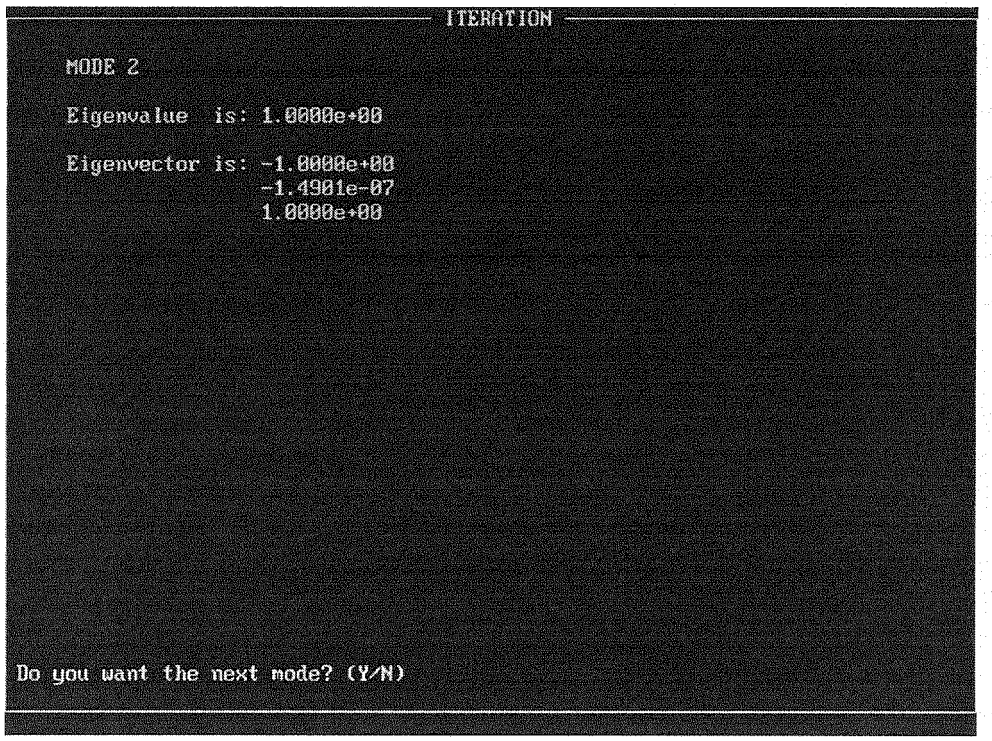
รูปที่ 43 แสดงการรับค่า input matrix [K] เพื่อใช้ในการคำนวณ
สำหรับโปรแกรม Iteration



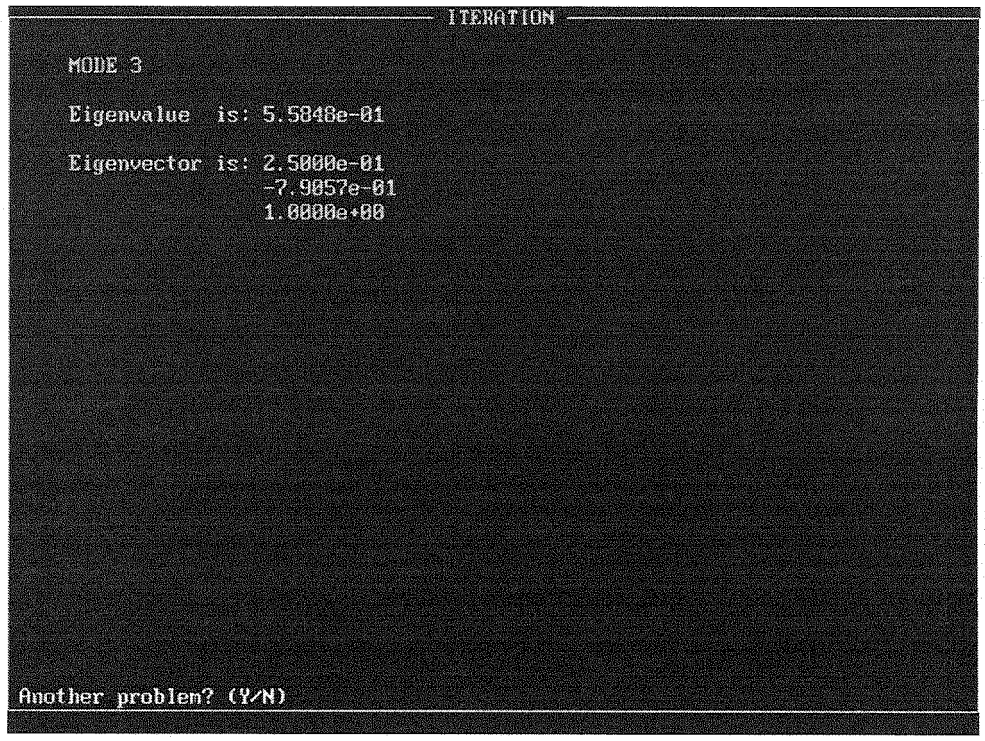
รูปที่ 44 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของค่า input matrix [K]
สำหรับโปรแกรม Iteration



รูปที่ 45 แสดงผลลัพธ์ค่า eigenvalue และ eigenvector
ชุดแรกที่ได้จากการคำนวณในโปรแกรม Iteration



รูปที่ 46 แสดงผลลัพธ์ค่า eigenvalue และ eigenvector ชุดสองที่ได้จากการคำนวณในโปรแกรม Iteration



รูปที่ 47 แสดงผลลัพธ์ค่า eigenvalue และ eigenvector ชุดสามที่ได้จากการคำนวณในโปรแกรม Iteration

6.6.3 ตัวอย่างโปรแกรม Iterate

ในการแก้ปัญหาของระบบ N-DOF เพื่อหาค่า eigenvalues และ eigenvectors นี้สามารถใช้โปรแกรม Iterate คำนวณได้ รูปที่ 40 แสดงหน้าจอแสดงวิธีการและข้อจำกัดของโปรแกรม ส่วนรูปที่ 41 - 47 (หน้า 71 - 74) แสดงการรับค่า แสดงตัวเลขที่ถูกป้อนเป็นข้อมูลเข้าและการแสดงผลลัพธ์ของการทำ iteration ชุดแรกและชุดถัดไป โดยในตัวอย่างกำหนดให้มีการแสดงค่า eigenvalues และ eigenvectors จำนวน 3 ชุด

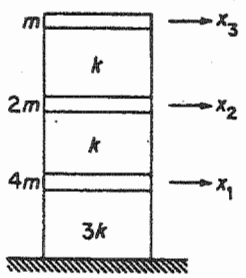
ตัวอย่างที่แสดงของโปรแกรม Iterate เป็นการแก้ปัญหาของสมการตัวอย่างที่กำหนดให้

$$\bar{\lambda} \quad = \quad k/m\omega^2 \qquad \text{แทนที่จะเป็น} \qquad \lambda \quad = \quad m\omega^2/k$$

จากสมการที่กำหนดให้ข้างต้น ค่า eigenvalue สำหรับโหมดแรก (โดยที่ mode shapes อื่นยังไม่เปลี่ยนแปลง) คือ

$$\lambda \quad = \quad 1/\bar{\lambda} \quad = \quad 1/4.775 \quad = \quad 0.2094$$

โจทย์ตัวอย่างนี้ได้มาจาก Thomson หน้า 238 - 240 และหน้า 243 -245 ซึ่งเป็นปัญหาเชิง flexibility ดังไดอะแกรมในรูปที่ 48 ที่ต้องการหาความถี่ธรรมชาติที่ต่ำ (น้อย) สุดโดยการทำ iteration



รูปที่ 48 ไดอะแกรมของระบบ 3-DOF ที่มี
ค่า mass และ stiffness ดังแสดงในรูป

จากการคำนวณหา by hand calculation ได้ค่าความถี่ต่ำสุด คือ

$$\omega_1 = \sqrt{\frac{3k}{14.32m}} = 0.457\sqrt{\frac{k}{m}}$$

โดยมี mode shape

$$\varnothing_1 = \begin{Bmatrix} 0.250 \\ 0.790 \\ 1.000 \end{Bmatrix}$$

จากนั้นก็นำค่า eigenvalue และ eigenvector ของโหมดแรกมาหาค่าโหมดที่สองและโหมดที่สาม ได้ผลสรุปของ solution ดังนี้

<u>โหมดแรก</u>

ค่า eigenvalue, $\bar{\lambda}_1 = 4.775 \qquad \lambda_1 = 0.2094$

$$\omega_1 = 0.4576\sqrt{\frac{k}{m}} \qquad \text{ค่า eigenvector, } \varnothing_1 = \begin{Bmatrix} 0.250 \\ 0.791 \\ 1.000 \end{Bmatrix}$$

<u>โหมดสอง</u>

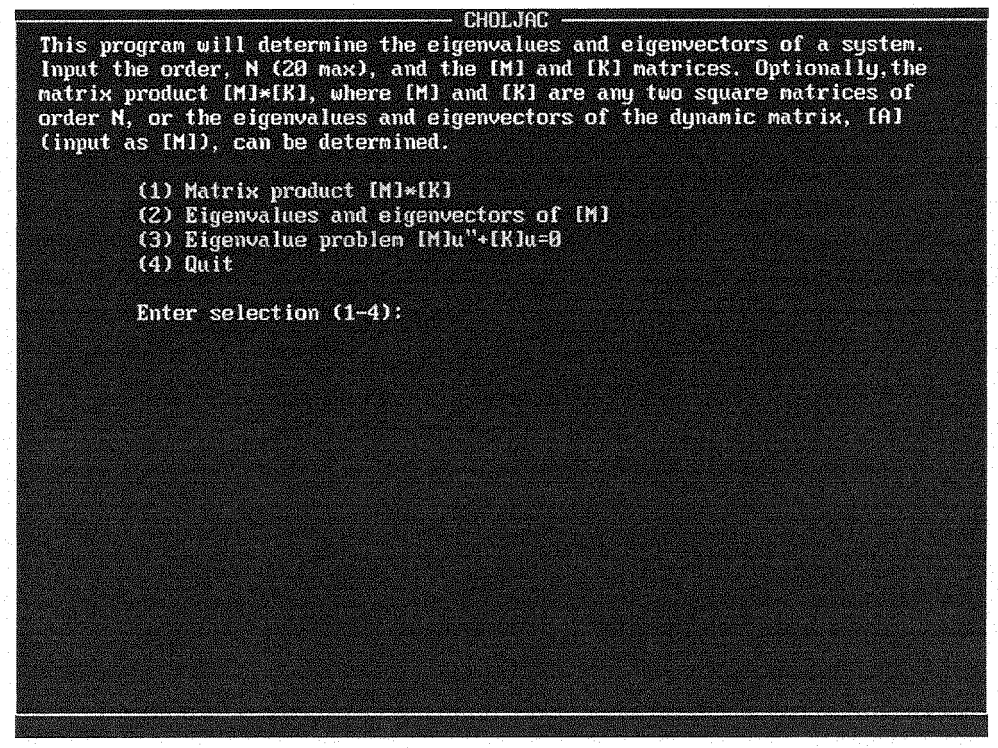
ค่า eigenvalue, $\bar{\lambda}_2 = 1.0000 \qquad \lambda_2 = 1.0000$

$$\omega_2 = 1.0000\sqrt{\frac{k}{m}} \qquad \text{ค่า eigenvector, } \varnothing_2 = \begin{Bmatrix} -1 \\ 0 \\ 1 \end{Bmatrix}$$

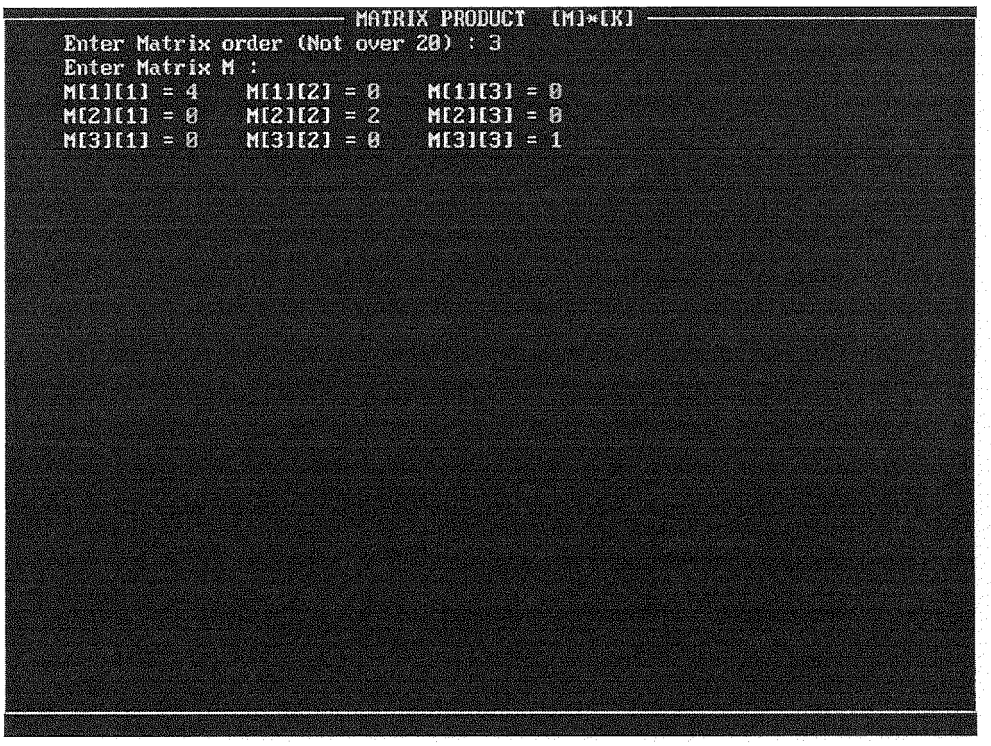
<u>โหมดสาม</u>

ค่า eigenvalue, $\bar{\lambda}_3 = 0.5585 \qquad \lambda_3 = 1.7906$

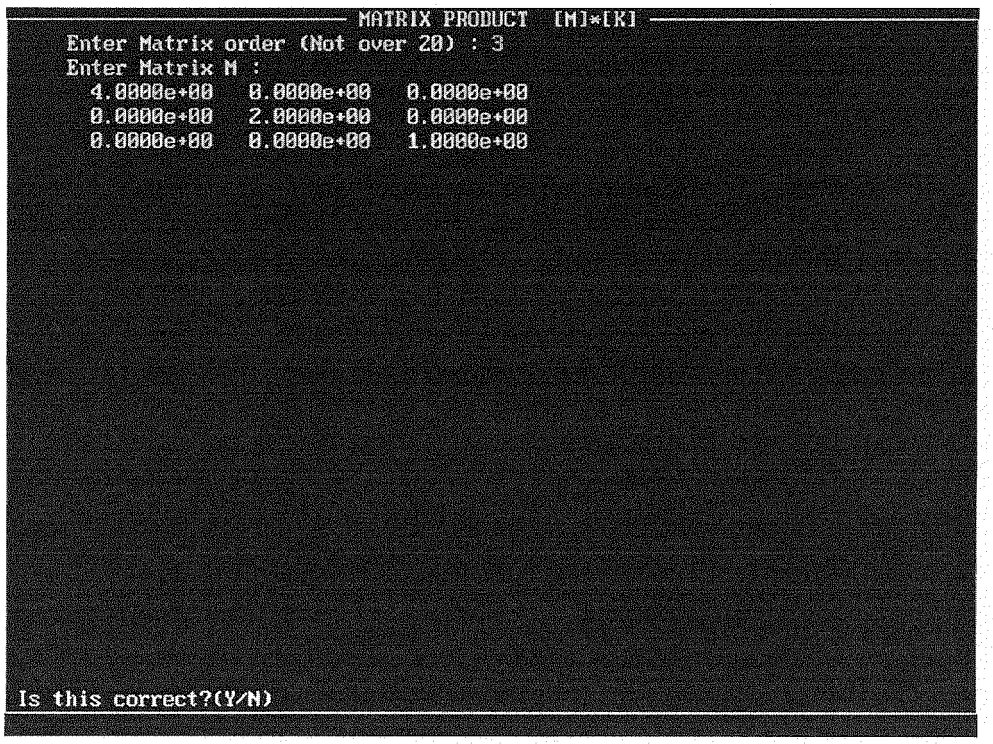
$$\omega_3 = 1.3381\sqrt{\frac{k}{m}} \qquad \text{ค่า eigenvector, } \varnothing_3 = \begin{Bmatrix} 0.250 \\ -0.791 \\ 1.000 \end{Bmatrix}$$



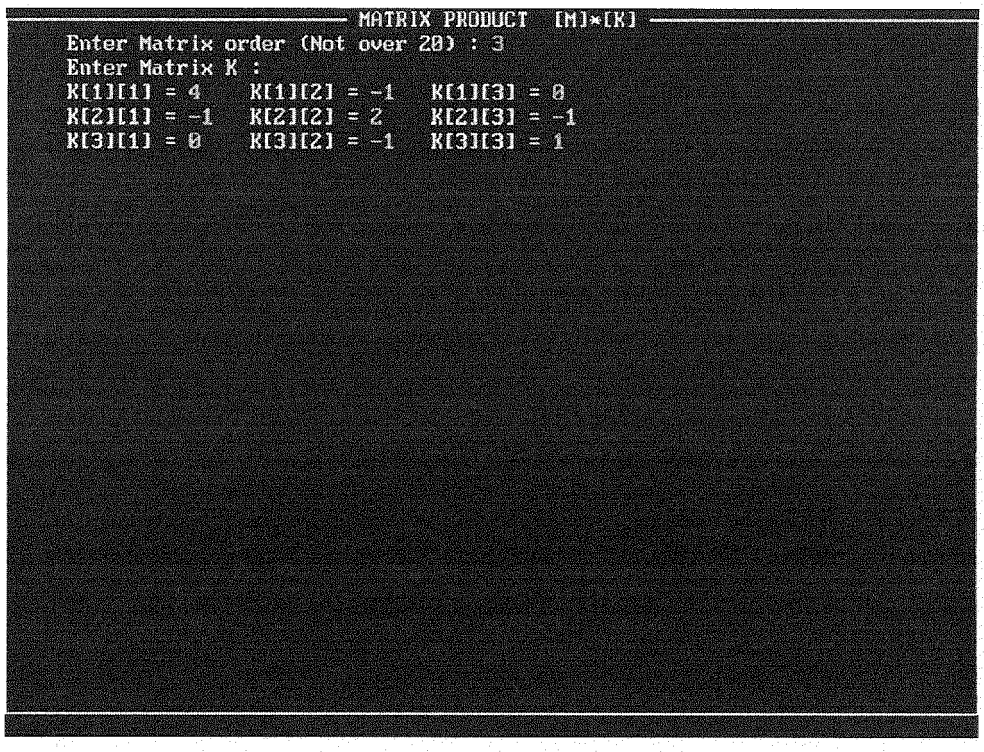
รูปที่ 49 เมนูของโปรแกรม Choljac แสดงขั้นตอนการคำนวณและข้อจำกัด



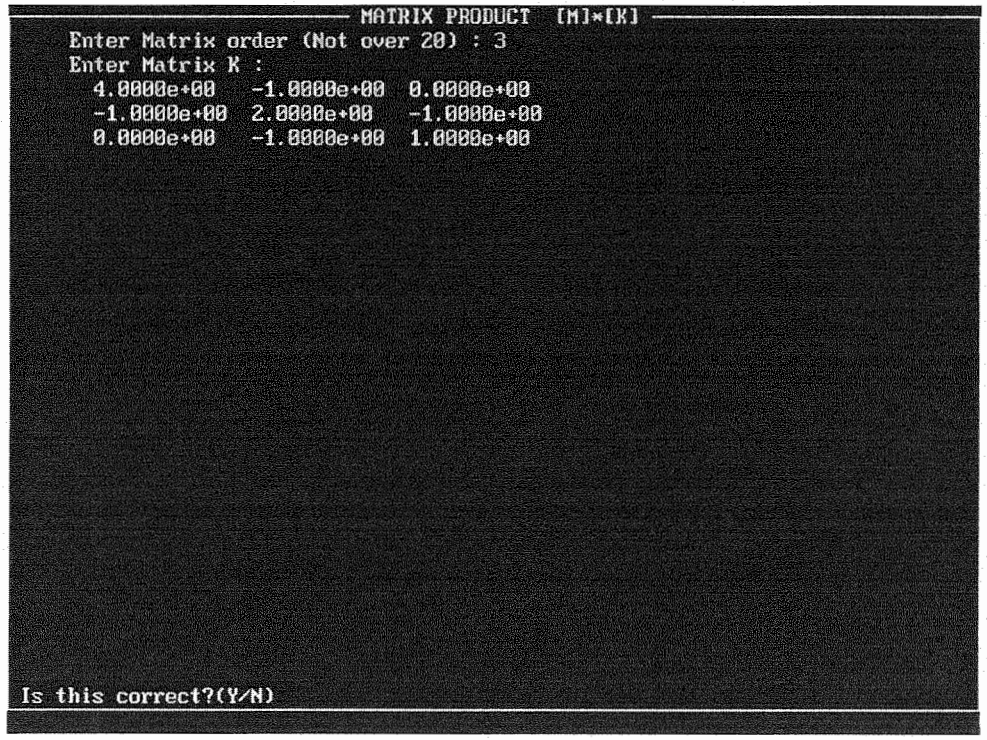
รูปที่ 50 แสดงการรับค่า input matrix [M] เพื่อหา
matrix product [M] * [K] ของโปรแกรม Choljac



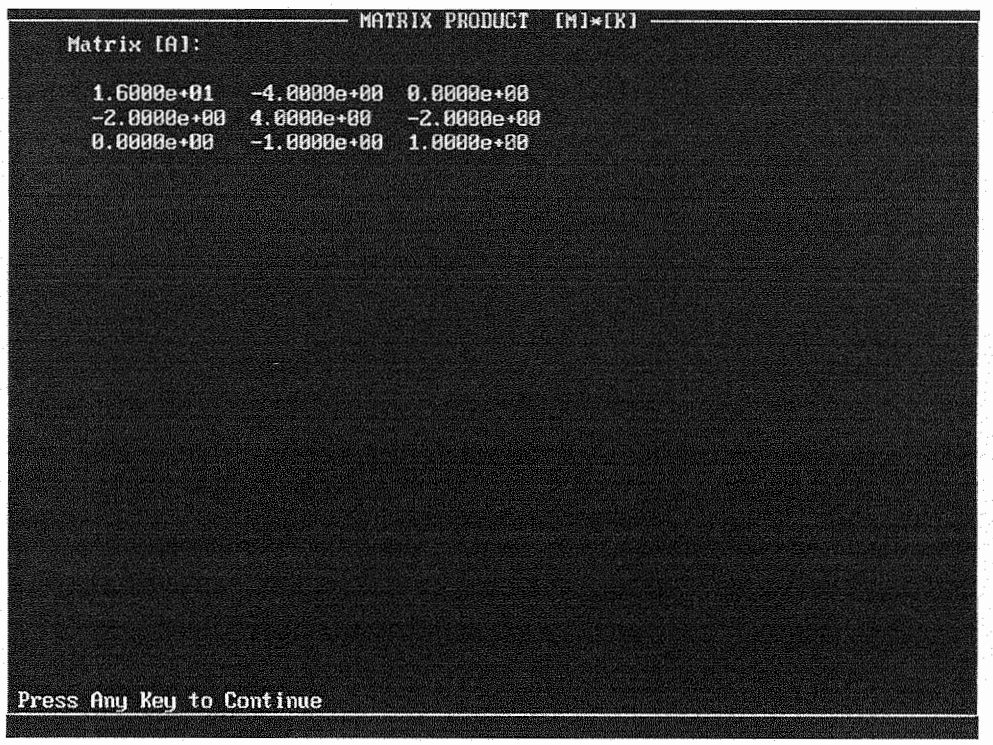
รูปที่ 51 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของ
ค่า input matrix [M] ในโปรแกรม Choljac



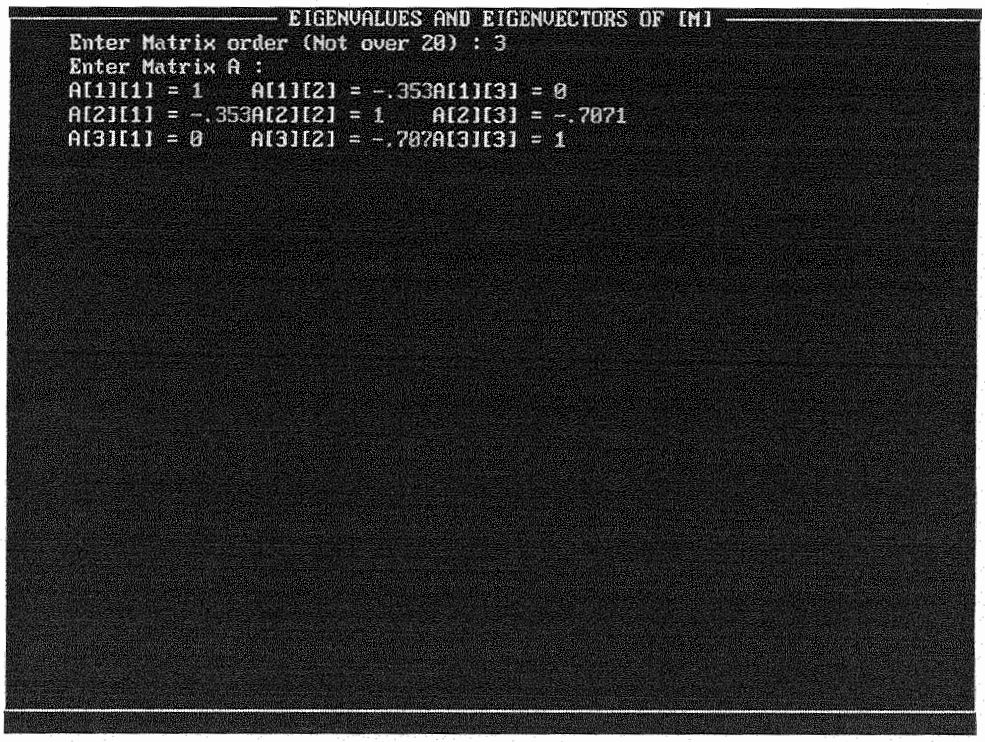
รูปที่ 52 แสดงการรับค่า input matrix [K] เพื่อหาค่า
matrix product [M] * [K] ในโปรแกรม Choljac



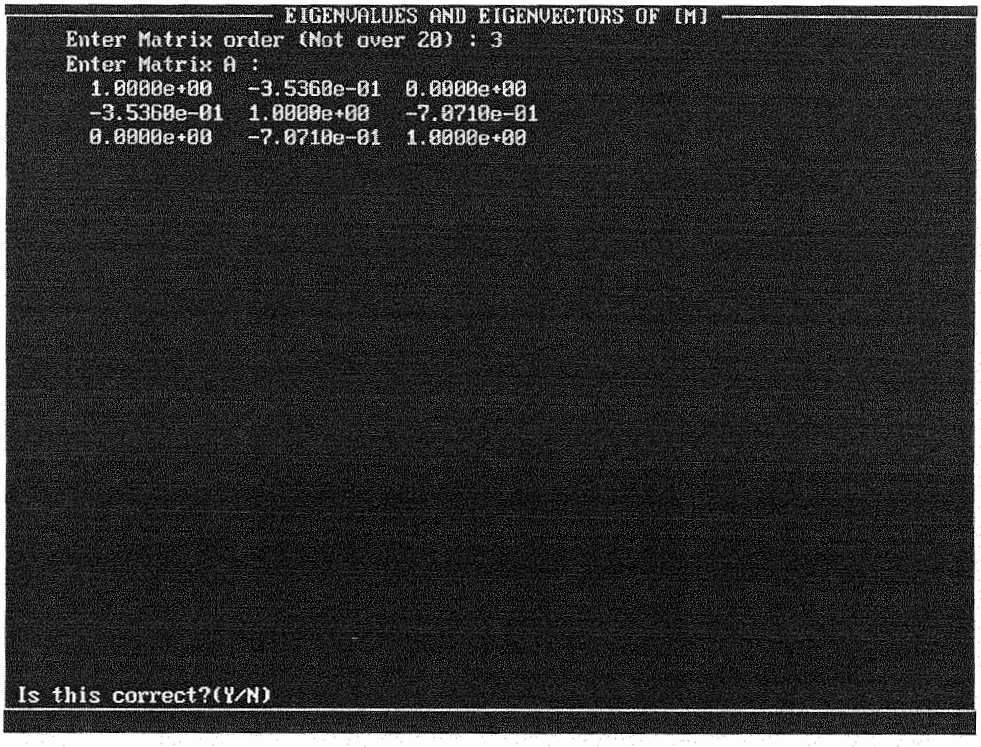
รูปที่ 53 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของค่า
input matrix [K] ของโปรแกรม Choljac



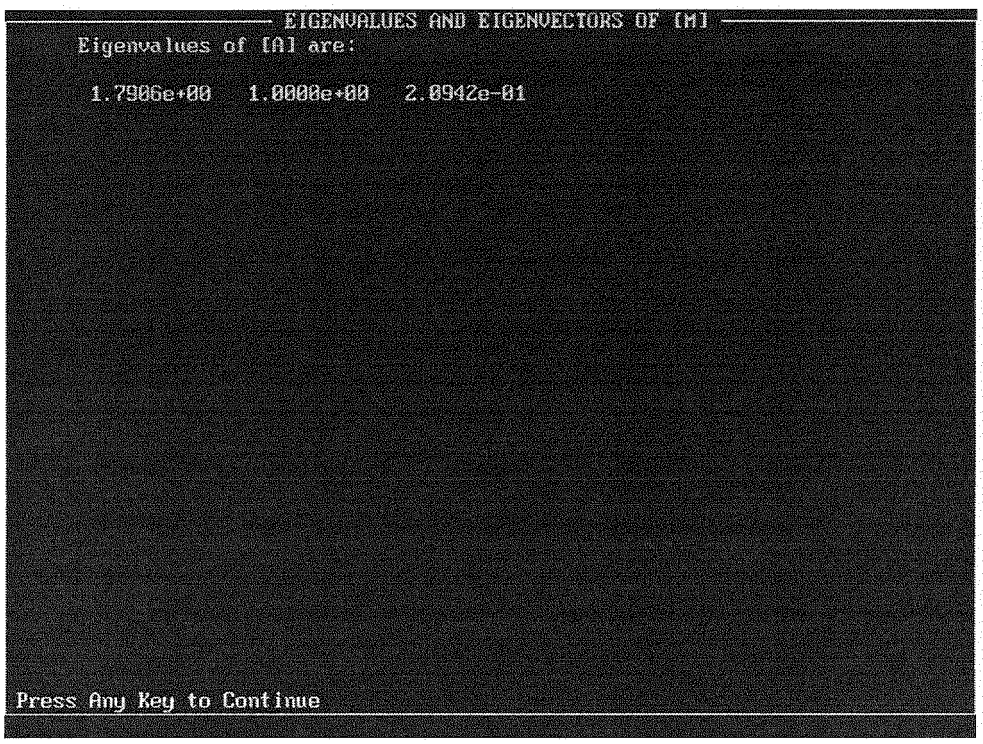
รูปที่ 54 แสดงผลลัพธ์ค่า matrix [A] ที่ได้จากคำนวณ
ค่า [M] * [K] ในโปรแกรม Choljac



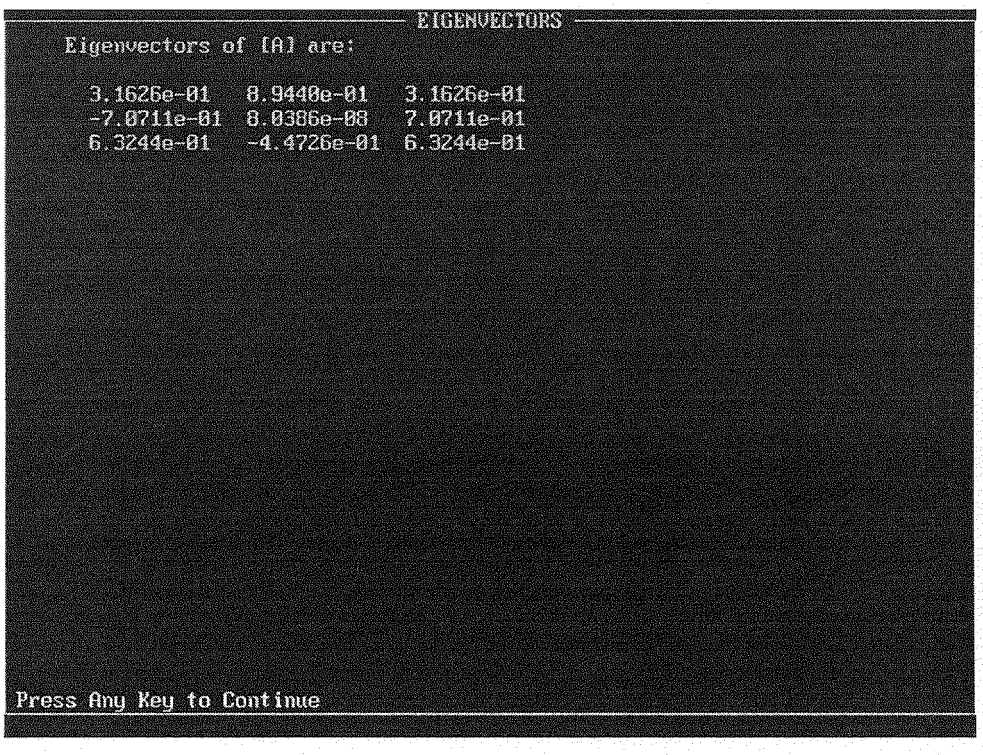
รูปที่ 55 แสดงการรับค่า input matrix [A] เพื่อหาค่า eigenvalues และ eigenvectors ในโปรแกรม Choljac



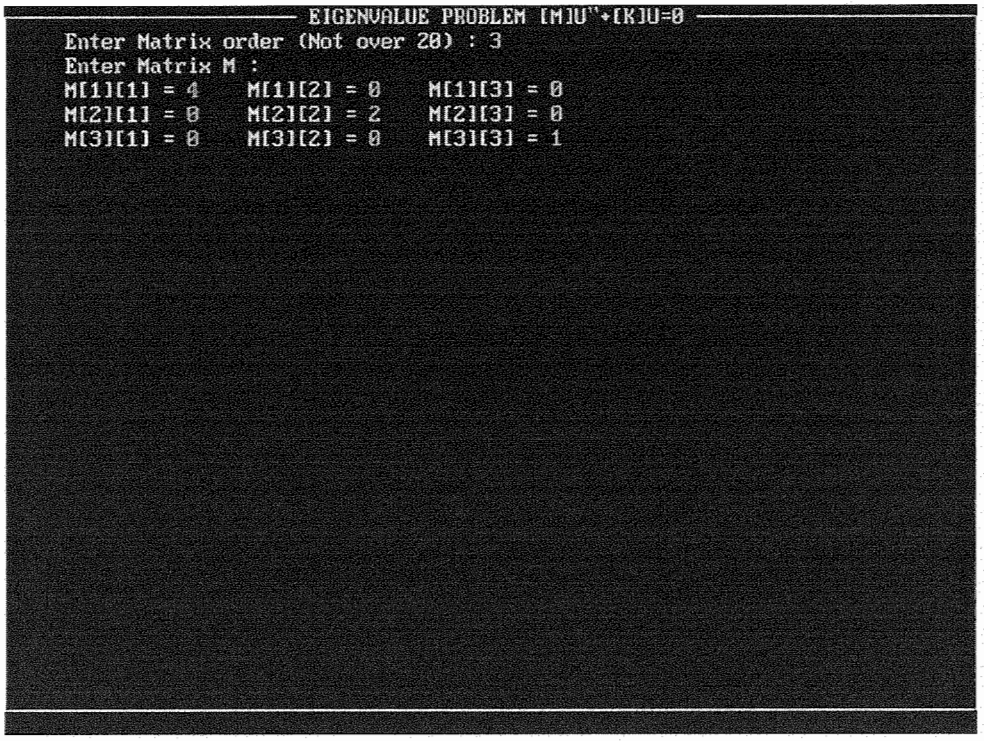
รูปที่ 56 แสดงตัวเลขเป็นทศนิยมสี่หลักของค่า input matrix [A] ในโปรแกรม Choljac



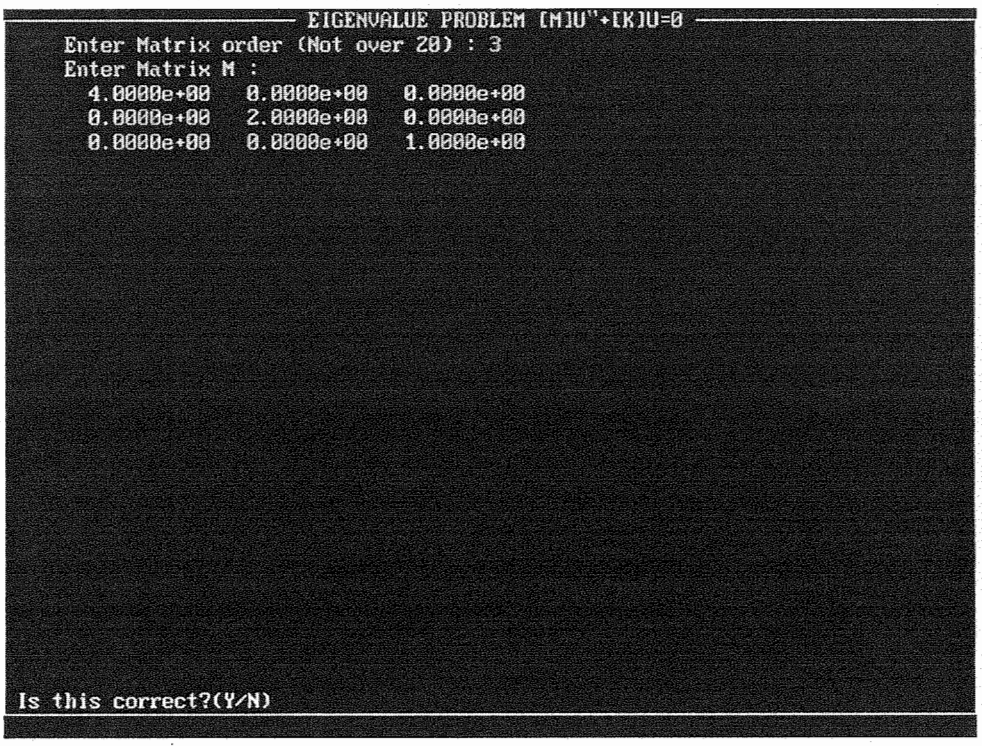
รูปที่ 57 แสดงผลลัพธ์ค่า eigenvalues ที่ได้จาก
ค่า input matrix [A] ในโปรแกรม Choljac



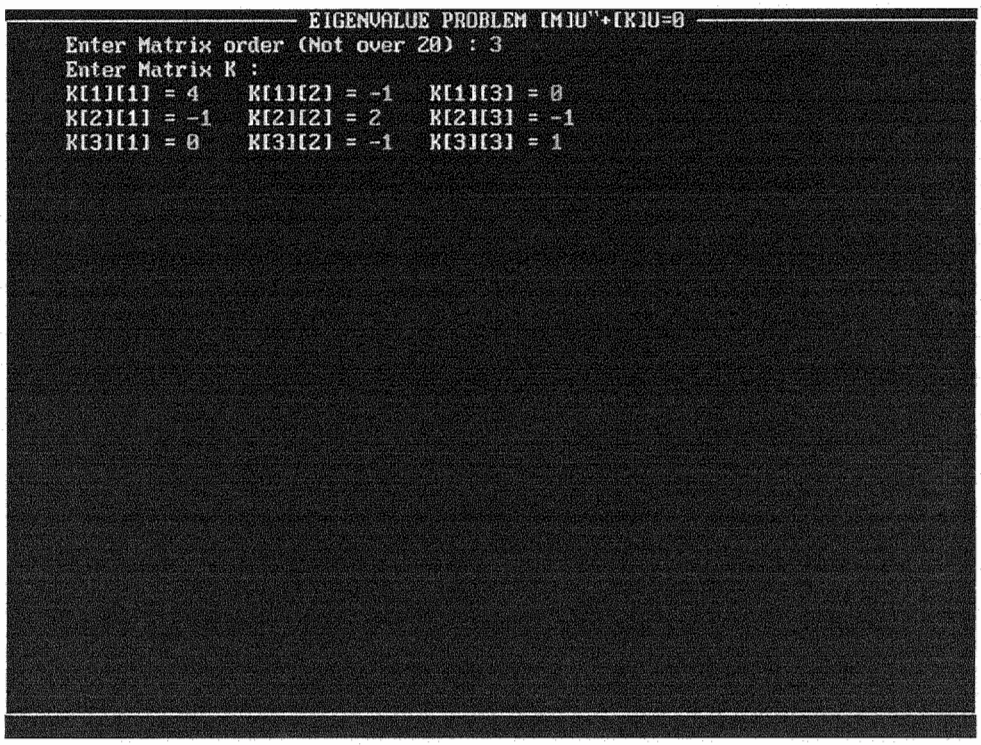
รูปที่ 58 แสดงผลลัพธ์ค่า eigenvectors ที่ได้จาก
ค่า input matrix [A] ในโปรแกรม Choljac



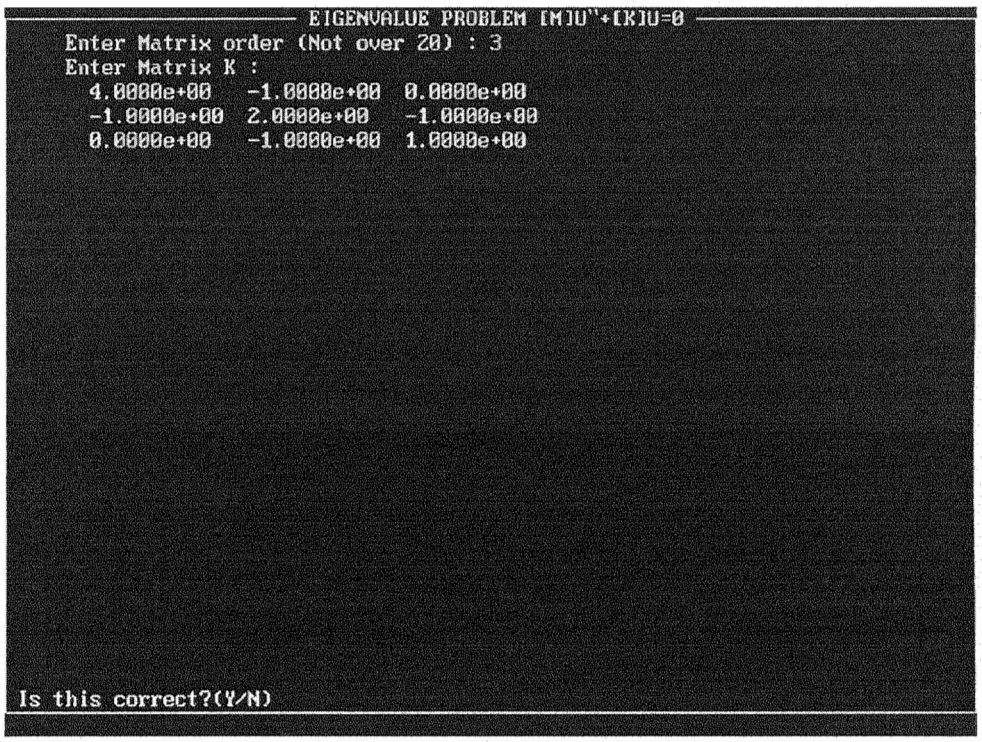
รูปที่ 59 แสดงการรับค่า input matrix [M] เพื่อหาค่า eigenvalues และ eigenvectors ของโปรแกรม Choljac



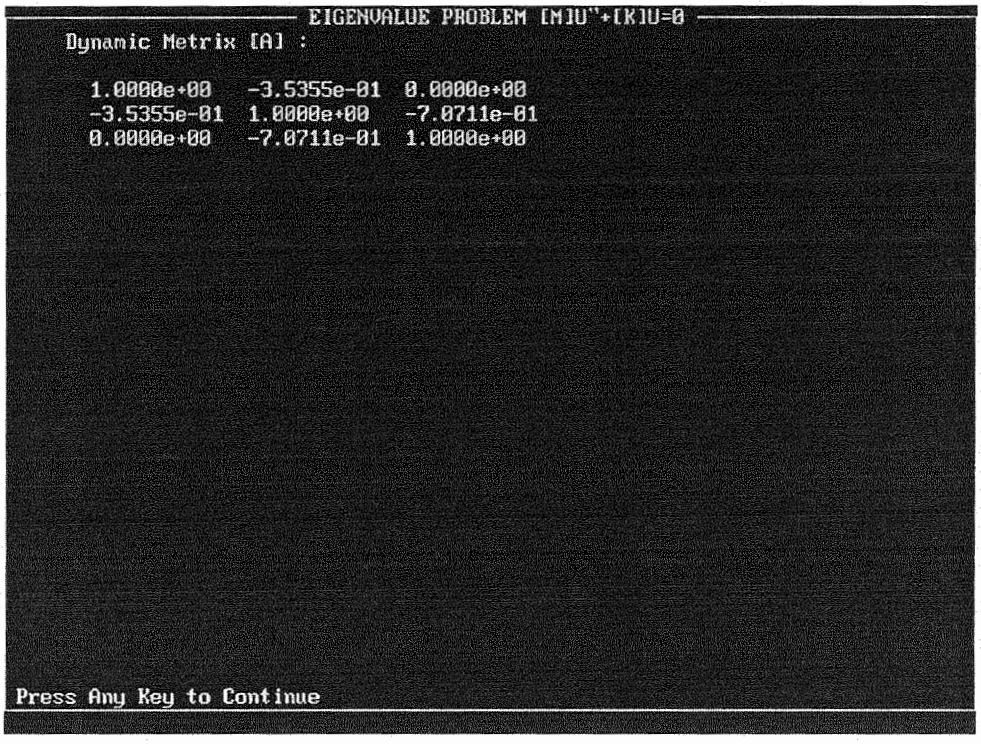
รูปที่ 60 แสดงตัวเลขเป็นทศนิยมสี่หลักของค่า input matrix [M] ในโปรแกรม Choljac



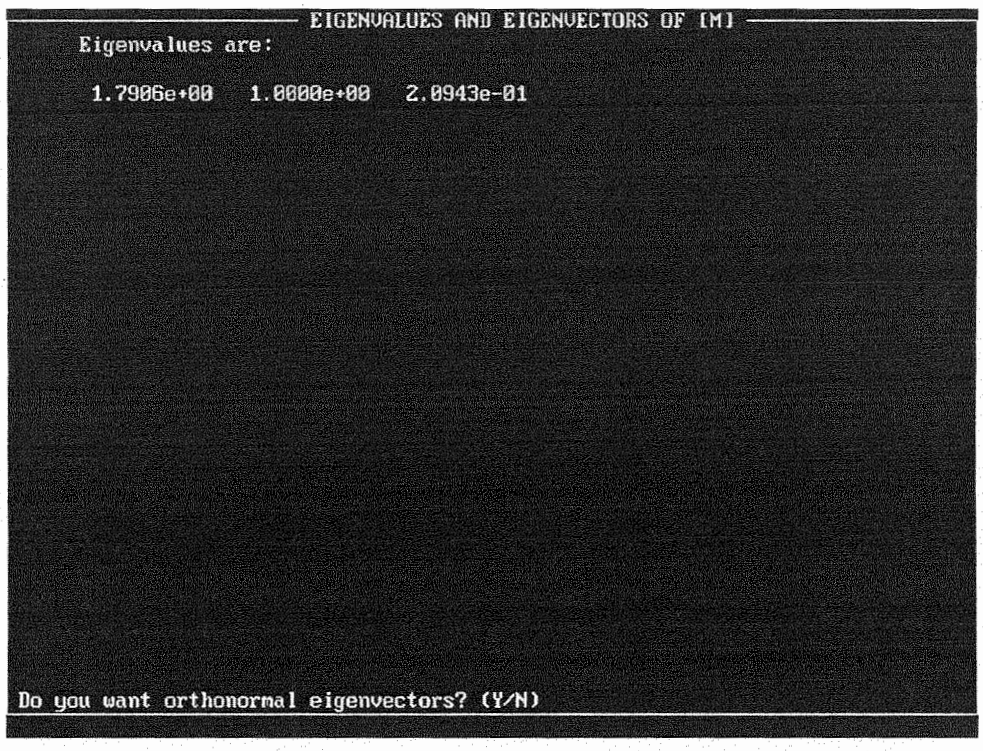
รูปที่ 61 แสดงการรับค่า input matrix [K] ที่ใช้ในการคำนวณหาค่า eigenvalues และ eigenvectors ในโปรแกรม Choljac



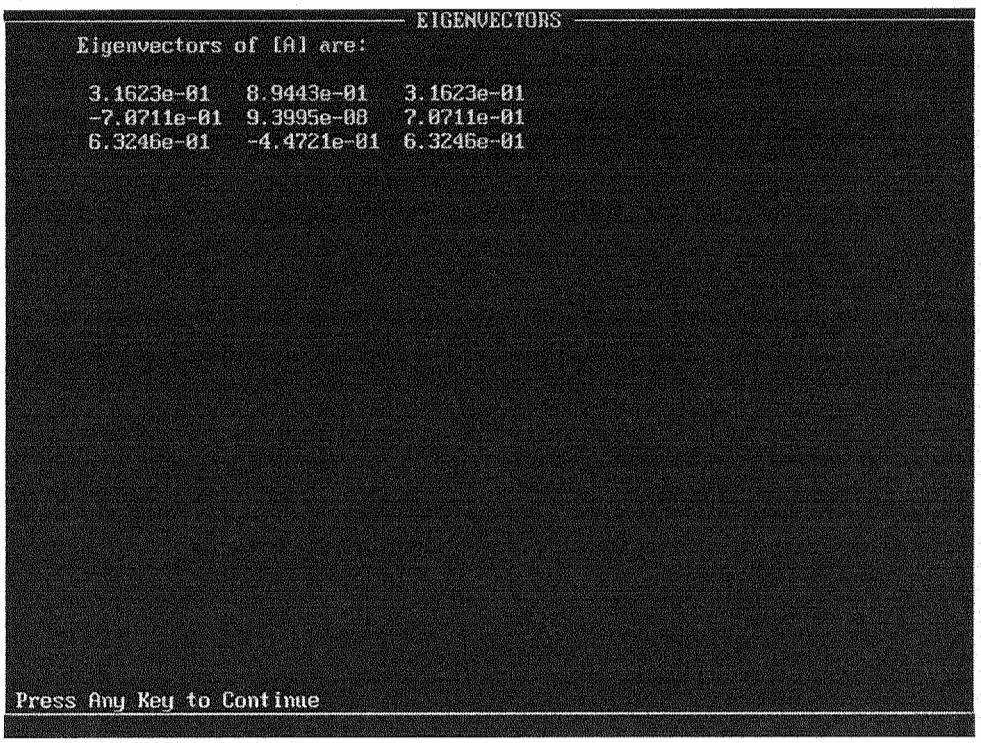
รูปที่ 62 แสดงค่าตัวเลขเป็นทศนิยมสี่หลักของ ค่า input matrix [K] ในโปรแกรม Choljac



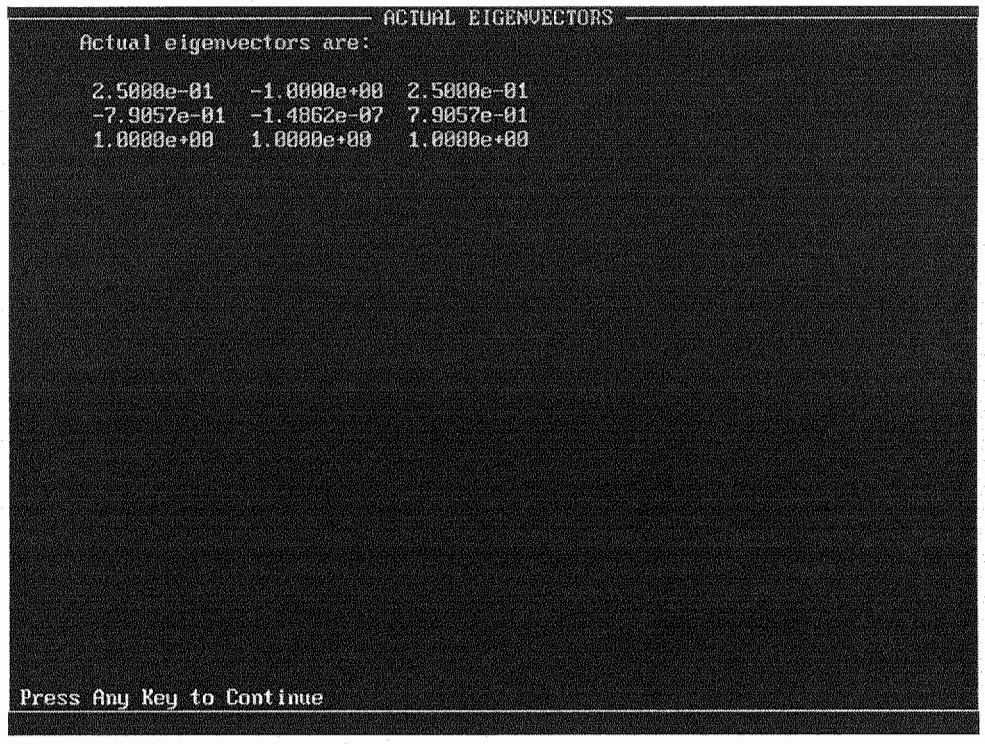
รูปที่ 63 แสดงผลลัพธ์ค่า dynamic matrix [A] ที่ใช้ Choleski decomposition และ Jacobi diagonalization เพื่อแก้ปัญหาของ eigenvalue problem



รูปที่ 64 แสดงผลลัพธ์ค่า eigenvalues ของ dynamic matrix [A] ที่คำนวณได้จากโปรแกรม Choljac



รูปที่ 65 แสดงผลลัพธ์ค่า eigenvectors ของ dynamic matrix [A]
ซึ่งคำนวณได้จากโปรแกรม Choljac



รูปที่ 66 แสดงผลลัพธ์ actual eigenvectors ที่คำนวณได้จากโปรแกรม Choljac

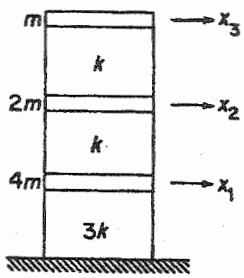
6.6.3 ตัวอย่างโปรแกรม Choljac

การแก้ปัญหาของระบบ N-DOF เพื่อหา matrix product, eigenvalue และ eigenvector โดยใช้วิธีการของ Choleski decomposition และ Jacobi diagonalization ในรูปที่ 49 ได้แสดงรูปแบบเมนูของการคำนวณในโปรแกรม Choljac นี้ ส่วนรูปที่ 50 - 66 (หน้าที่ 77 - 85) แสดงการรับค่า แสดงผลคูณของเมตริกซ์ ตลอดจนแสดงขั้นตอนการรับค่าซ้ำที่ใช้ในการหา eigenvalues และ eigenvectors

โจทย์ตัวอย่างนี้ได้มาจาก Thomson หน้า 251 - 252 และหน้า 255 - 258 ซึ่งเป็นปัญหาของแบบจำลองของ 3-DOF ที่มีสมการการเคลื่อนที่ที่กำหนดและต้องการ decompose ค่า stiffness matrix แล้วหาค่า eigenvalues และ eigenvectors ของระบบ

รูปที่ 67 ข้างล่างนี้เป็นอาคารสิ่งก่อสร้างที่มี 3-DOF และมีค่าสมการการเคลื่อนที่เป็น

$$\left[-\left(\frac{\omega^2 m}{k}\right)\begin{bmatrix}4 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 1\end{bmatrix}+\begin{bmatrix}4 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 1\end{bmatrix}\right]\begin{Bmatrix}x_1\\ x_2\\ x_3\end{Bmatrix}=\begin{Bmatrix}0\\ 0\\ 0\end{Bmatrix}$$



รูปที่ 67 แบบจำลอง 3-DOF ของอาคารสิ่งก่อสร้างที่มีสมการการเคลื่อนที่ที่กำหนดและต้องการ decomposing หารากสมการและค่า eigenvectors

จากการทำ matrix transformation ทำให้สามารถ decomposing stiffness matrix เป็นรูปแบบมาตรฐาน คือ

$$[-\bar{\lambda} I + \tilde{A}]\gamma = 0$$

โดยที่ $\bar{\lambda} = k/\omega^2 m$

$X = U^{-1}Y$

ดังนั้นเมื่อ mass matrix ในรูปที่ 67 ถูก decomposed จะได้รูปแบบมาตรฐานของสมการการเคลื่อนที่ได้เป็น

$$\left[-\lambda I + \begin{bmatrix} 1.0 & -0.3536 & 0 \\ -0.3536 & 1.0 & -0.7071 \\ 0 & -0.7071 & 1.0 \end{bmatrix}\right]\begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

โดยที่ $\lambda = \omega^2 m/k$

สำหรับการแก้ปัญหาต่อไปก็ใช้วิธี Jacobi ทำการ diagonalize ค่าของ dynamic matrix ซึ่งมีค่าเป็น

$$\tilde{A} = \left[\begin{array}{c|cc} 1.0 & -0.3536 & 0 \\ \hline -0.3536 & 1.0 & -0.7071 \\ 0 & -0.7071 & 1.0 \end{array}\right]$$

จากนั้นก็ลดขนาดของ off-diagonal terms โดยการทำกระบวนการซ้ำๆ ค่า eigenvalues ได้มาจาก diagonal elements ของ $\tilde{A}$ และค่า eigenvectors ของ $\tilde{A}$ เป็นผลคูณมาจากค่าของ rotation matrices, R_i เมื่อกำหนดให้ l ที่เป็น subscript ของ iteration ครั้งสุดท้ายได้ความสัมพันธ์ของ dynamic matrix และ rotational matrix เป็นไปตามสมการข้างล่างคือ

$$\tilde{A}_l = R_l^T \cdots R_k^T R_{k-1}^T \cdots R_2^T R_1^T \left[\tilde{A}_1\right] R_1 R_2 \cdots R_{k-1} R_k \cdots R_l = \Lambda$$

$$\lim_{l\to\infty} R_1 R_2 \cdots R_l = \tilde{P}$$

ต่อมาค่า eigenvectors เหล่านี้ถูกเปลี่ยนเป็น transformed equation ในจุดพิกัด y และตัวถูกเปลี่ยนกลับ (convert) ไปเป็น eigenvectors ของสมการดั้งเดิมในจุดพิกัด x โดยใช้สมการที่ 83 หัวข้อ 5.3.6 หน้า 48 นั่นคือ

$$X = U^{-1}Y$$

เมื่อเปรียบเทียบค่า eigenvalues ที่ได้จาก symmetry dynamic matrix, $\tilde{A}_3$ กับที่คำนวณด้วยโปรแกรมคอมพิวเตอร์มีผลดังนี้

ผลจาก $\tilde{A}_3$	ผลจากโปรแกรม
$\lambda_1 = 0.213$	$\lambda_1 = 0.2094$
$\lambda_2 = 1.014$	$\lambda_2 = 1.0000$
$\lambda_3 = 1.817$	$\lambda_3 = 1.7905$

สำหรับค่า eigenvectors จะได้ค่า

$$Y = R_1 R_2 R_3 = \begin{bmatrix} 0.9011 & 0.3029 & 0.3102 \\ 0.0276 & 0.6739 & -0.7383 \\ -0.327 & 0.6739 & 0.5988 \end{bmatrix}$$

$$X = U^{-1}Y = \begin{bmatrix} 0.006 & 0.1515 & 0.1551 \\ 0.0195 & 0.765 & -0.5221 \\ -0.327 & 0.6739 & 0.5988 \end{bmatrix}$$

ซึ่งเมื่อถูก normalized ด้วย 1.0 จะได้ผลเปรียบเทียบดังนี้

$$X = \begin{bmatrix} -0.90 & 0.225 & 0.259 \\ -0.05 & 0.707 & -0.872 \\ 1.00 & 1.00 & 1.00 \end{bmatrix} \quad \text{จาก } \tilde{A}_3$$

mode2 mode1 mode3

$$X = \begin{bmatrix} -1.00 & 0.25 & 0.25 \\ 0 & 0.79 & -0.79 \\ 1.00 & 1.00 & 1.00 \end{bmatrix}$$ จากโปรแกรมคอมพิวเตอร์

6.7 Source Codes ของโปรแกรมแบบจำลองคลื่นการสั่นสะเทือน

ในภาคผนวกได้พิมพ์ source codes ของ โปรแกรมต่างๆ ที่ประกอบเป็นโปรแกรม Vib.Exe โดยแบ่งออกได้ดังนี้

1. ภาคผนวก ก. แสดง source code ของโปรแกรม Vib.Cpp ซึ่งเป็นโปรแกรมที่เขียนขึ้นเพื่อช่วยในการจัดเมนูของหน้าจอคอมพิวเตอร์

2. ภาคผนวก ข. แสดง source code ของโปรแกรม Kutta. Cpp ซึ่งเป็นโปรแกรมที่เขียนเพื่อใช้ในการคำนวณตามวิธีของ Runge-Kutta ในระบบ SDF

3. ภาคผนวก ค. แสดง source code ของโปรแกรม Poly. Cpp ซึ่งเป็นโปรแกรมที่เขียนเพื่อใช้ในการคำนวณ polynomial equation โดยใช้วิธี Gauss elimination ในระบบ N-DOF

4. ภาคผนวก ง. แสดง source code ของโปรแกรม Iterate. Cpp ซึ่งโปรแกรมทำ matrix iteration กับ matrix deflation ตามวิธีของ Choleski กับวิธีของ Gram-Schmidt ในระบบ N-DOF

5. ภาคผนวก จ. แสดง source code โปรแกรม Choljac. Cpp ซึ่งเป็นโปรแกรมหาผลคูณของแมตริกซ์ หาค่า eigenvalues และ eigenvectors ตามวิธีของ Choleski กับวิธีของ Jacobi

บทที่ 7

บทวิจารณ์และบทสรุปของการพัฒนาแบบจำลอง

งานวิจัยเรื่อง "การพัฒนาแบบจำลองการสั่นสะเทือนเนื่องจากการระเบิด" ที่ได้ดำเนินการมาในช่วงปีแรก ได้มีการวิจารณ์และสรุปผลงานของงานวิจัยชิ้นนี้ตามลำดับดังนี้

7.1 การศึกษาถึงคุณสมบัติคลื่นสั่นสะเทือน

คลื่นที่ เป็นผลมาจากการระเบิดเป็นคลื่นพัลส์ที่ถูกกระตุ้นอย่างทันทีทันใด มีการตอบสนองต่อแรงที่มากระทำชั่วครู่ (transient response) มีความเร็วในการเดินทางผ่านตัวกลางสูงมาก อยู่ในรูปแบบของคลื่นกระแทก (impact wave)

ในเชิงทฤษฎีได้มีการศึกษาไว้แล้วในเรื่องการกระตุ้นของอิมพัลส์ ดังในรายละเอียดที่อยู่ในบทที่ 3 และบทที่ 4 ของรายงานฉบับนี้มีหัวข้อพอสรุปเป็นประเด็นหลักได้ดังนี้

1. อิมพัลส์ที่เกิดจากการระเบิดเป็นแรงกระแทกที่มีขนาดใหญ่มากกระทำในช่วงเวลาสั้นๆ

2. Time integral ของแรงอิมพัลส์มีค่าเป็น finite ในเชิงคณิตศาสตร์จะแทนหน่วยอิมพัลส์ (impulse unit) ได้เท่ากับ delta function

3. เมื่อวิเคราะห์ถึงระบบที่จะเลือกใช้ในการแก้ปัญหาคลื่นกระแทก จำเป็นต้องคำนึงถึงผลของความเร็ว (velocity) กับความเสียดทาน (friction) เข้ามาเกี่ยวข้อง จึงต้องใช้ viscously damped vibration system ในการวิเคราะห์เพื่อแก้ปัญหา

7.2 แนวทางในการแก้ปัญหาในปัจจุบัน

วิธีการแก้ปัญหาของการสั่นสะเทือนนี้จะเริ่มต้นการวิเคราะห์ปัญหาจากจุดง่ายไปยังจุดที่ยุ่งยาก งานวิจัยที่นำเสนอมาในรายงานความก้าวหน้าครั้งที่ 1 และในรายงานฉบับสมบรูณ์ปีแรกนี้จึงมีแนวทางที่ได้เสนอไว้สำหรับรายงานปีแรก ดังนี้

1. วิธีการเริ่มต้นตั้งข้อจำกัดในเรื่องของการสั่นสะเทือนเนื่องจากการระเบิดเป็นระบบ Single Degree of Freedom กำหนดให้แรงที่กระทำภายนอกแปรผันกับเวลาและทิศทางของการเคลื่อนที่ โดยสรุปว่าการไหวสะเทือนของพื้นผิวดินมีผลต่อพฤติกรรมของระบบและแสดงออกในลักษณะของ response spectra

2. แบบจำลองคลื่นการสั่นสะเทือนที่ได้เสนอเป็นแนวทางแรก คือ เป็น viscously damped free vibration ซึ่งใช้การแก้ปัญหาด้วย differential equation of motion ซึ่งแสดงในเทอมของ critical damping fraction (β) และ natural circular frequency ได้เป็น

$$\ddot{x} + 2\beta\omega\dot{x} + \omega_n^2 x = \frac{F}{m}$$

เมื่อค่า x เป็นค่าการเปลี่ยนตำแหน่งสมบรูณ์ของพื้นผิวดิน (absolute ground displacement) ค่า F เป็นแรงกระทำภายนอกและ m เป็นมวลสาร

3. สมการของแบบจำลองข้อ 2 สามารถนำมาดัดแปลงเป็นสมการเคลื่อนที่เชิงพลศาสตร์พื้นฐานในรูปแบบการเปลี่ยนตำแหน่งสัมพันธ์ สำหรับกรณีของระบบ SDF โดยเขียนในแบบฟอร์มใหม่ คือ

$$\ddot{\delta} + 2\beta\omega_n\dot{\delta} + \omega_n^2 = -\ddot{u}$$

โดยที่

δ = relative displacement = x - u

u = absolute displacement of the ground

สมการแบบฟอร์มใหม่นี้เป็นผลมาจากการกระตุ้นจากแรงภายนอก (external force-excited system) สามารถนำมาใช้กับการเคลื่อนที่ของระบบกระตุ้นพื้นฐาน (base-excited system) เมื่อค่า F/m ถูกแทนที่ด้วย x หรือเป็นค่าลบของความเร่งพื้นฐาน (base acceleration)

4. การวิเคราะห์ปัญหาเชิง numerical เพื่อแก้ปัญหาของ differential equation ที่กล่าวมาในหัวข้อ 3 สามารถแทนได้ด้วยรูปฟอร์มทั่วไปเป็น

$$\ddot{x} = f(x, \dot{x}, t)$$

$$x_0 = x(0)$$

$$\dot{x}_0 = \dot{x}(0)$$

โดยกำหนดให้ค่าสภาวะเริ่มต้น x_0 และ x_0 โดยให้มีค่าตามธรรมชาติ, τ_n เป็น $2\pi (m/k)^{\frac{1}{2}}$ นอกจากนี้ยังจะต้องทราบค่าและเป็นการเริ่มต้นที่เวลา t = o

ต่อมาก็นำวิธีการของ Runge-Kutta ที่ได้จากทำการ expansion ของค่าฟังก์ชัน x = F (t) และค่าของ derivatives ของ x โดยใช้ Taylor's series

หลักการสำคัญของวิธี Runge-Kutta คือ การที่ลดค่าสมการเดี่ยวของ second-order differential equation ไปเป็นสองสมการของ first-order differential equation

จากรูปแบบของการคาดคะเนการสั่นสะเทือนโดย dynamical differential equation ของระบบ SDF

$$m\ddot{x} + c_1\dot{x} + kx = f(t)$$

เมื่อทำการสมมุติค่าให้ y = x ก็สามารถเขียนสมการในรูปแบบที่สามารถนำ คอมพิวเตอร์มาประยุกต์ใช้เป็น

$$\dot{y} = \frac{1}{m}\left[f(t) - c_1 y - kx\right] = F(x, y, t)$$

ซึ่งค่าพารามิเตอร์ของ m (mass), c_1 (damping coefficient) และ k (spring stiffness) ทั้ง 3 พารามิเตอร์นี้จัดเป็น known numerical values เมื่อเทียบกับ time increment (Δt)

5. โปรแกรมแบบจำลองคลื่นการสั่นสะเทือนที่พัฒนาขึ้นมีชื่อโปรแกรมว่า "Kutta" เป็นโปรแกรมที่สร้างขึ้นมา สำหรับการสั่นสะเทือนที่เป็น damping system ในระบบ SDF เขียนขึ้นมาเพื่อใช้ภายใต้ Microsoft Windows version 3.1 Thai Edition ในช่วงการคำนวณจะทำใต้โปรแกรม C^{++} ของบริษัท Borland

ในโปรแกรมกำหนดค่าช่วงเวลาไว้สูงสุด 80 ค่า ผลลัพธ์ของโปรแกรมจะระบุค่าการเปลี่ยนตำแหน่งและความเร็วเมื่อเทียบกับเวลาแล้วพลอตออกมาเป็นกราฟ และยังพิมพ์ค่าคาบธรรมชาติ และค่าแอมพลิจูดสูงสุดไว้ด้วย

6. จากงานวิจัยที่ผ่านมาในเรื่องการตอบสนองของอาคารโครงสร้างต่อการสั่นสะเทือน (Hudson and Housner, 1957 ; Dowding, 1971) มีผลไปในทิศทางเดียวกันว่า พฤติกรรมของอาคารที่อยู่อาศัยที่มีความสูงไม่เกิน 3 ชั้น พบว่าจะมีความคล้ายคลึงกับระบบ Single Degree of Freedom แต่ถ้าอาคารโครงสร้างที่มีความสูงเกิน 3 ชั้น ก็จำเป็นต้องใช้แบบจำลองของระบบ Multidegree of Freedom

7. แนวทางในเรื่อง Multidegree of Freedom ในรายงานการวิจัยปีแรกนี้มุ่งประเด็นที่จะแก้ระบบที่มีจุดพิกัดที่ไม่พึ่งพิง (independent coordinates) มากกว่า 1 จุด หรือกำหนดให้ระบบมีความถี่เป็น N natural frequencies

ปัญหาในเรื่อง Multidegree of Freedom จึงต้องหาค่าของ normal mode ซึ่งค่า Quantities ในทางคณิตศาสตร์ที่เกี่ยวข้องกับเรื่องนี้ ได้แก่ ค่า eigenvalues และ eigenvectors

8. ในระบบ N-DOF การแก้ปัญหาจะยุ่งยากมาก มีวิธีการหลายวิธีการที่ถูกนำเสนอขึ้นมาเพื่อหาค่า eigenvalues และ eigenvectors ของสมการการเคลื่อนที่ คณะผู้วิจัยได้เสนอไว้ 3 รูปแบบ คือ

ก) การแก้ปัญหาของ Polynomial equation ด้วยวิธีของ Gauss เพื่อลดจำนวน matrix equation ทำให้ค่า eigenvectors หาได้ง่ายขึ้น

โปรแกรมแบบจำลองชื่อ "Poly" ที่พัฒนาขึ้นมาสำหรับวิธีหา coefficients ของ Polynomial equation กับนำค่าที่ได้ไปหา eigenvectors โดยใช้ Gaussian elimination เป็นแนวทางหนึ่งในการหา quantities ของระบบ N-DOF

ข) การแก้ปัญหาอีกแนวทางหนึ่งของระบบ N-DOF ที่มีค่าความถี่ต่ำสุดเพียง 2-3 ค่าและรูปร่างของโหมดที่มีไม่กี่โหมดต้องการหา การนำวิธีของ power method หรือที่นิยมเรียกกันว่าเป็น iteration method จะช่วยให้การแก้ปัญหาของ eigenvalue problem เป็นไปด้วยความรวดเร็ว

โปรแกรมแบบจำลองชื่อ "Iterate" เป็นการพัฒนาขึ้นมาสำหรับใช้สำหรับวิธี iteration โดยการคำนวณซ้ำๆกันเพื่อให้หาค่าความถี่ของโหมดต่ำสุดที่อยู่ใน toterance limit แล้วจึงนำมาใช้หาค่า eigenvectors ของ mode shape

ค) การแก้ปัญหาแนวทางสุดท้ายของระบบ N-DOF เป็นการ decomposing ค่า stiffness matrix และ diagonalize ค่า dynamic matrix ด้วยวิธีการของ Choleski และ Jacobi ซึ่งช่วยได้ดีสำหรับกรณีที่ระบบมี mass และ stiffness matrix ขนาดใหญ่

โปรแกรมแบบจำลองชื่อ "Choljac" ได้พัฒนาขึ้นมาสำหรับแก้ปัญหาระบบ N-DOF ที่มีการคำนวณหลายขั้นตอนและยุ่งยาก โดยสามารถใช้ได้กับขนาดของแมตริกซ์ที่ใหญ่ ซึ่งในการคำนวณจะใช้การ decomposing และ transformation เข้ามาช่วยในการหาค่า eigenvalues และ eigenvectors ที่เหมาะสม

7.3 แนวทางในการแก้ปัญหาในปัจจุบัน

งานวิจัยในเรื่อง "การพัฒนาแบบจำลองการสั่นสะเทือนเนื่องจากการระเบิด" จัดยังว่าไม่เสร็จสิ้นสมบรูณ์ เนื่องจากยังขาดการทดสอบข้อมูลและแนวทางแก้ปัญหายังไม่สมบรูณ์ต้องมีการพัฒนาต่อไปอีก ดังนี้

1. ควรมีการหา solutions ของ transient response ใน damped system เพิ่มเติมโดยเน้นเกี่ยวกับแรงภายนอกที่กระทำเป็นแบบ step function หรือ exponential decay type ซึ่งจะใกล้เคียงกับกระบวนการระเบิดหินในธรรมชาติ

การแก้ปัญหาในเรื่อง numerical solutions ของ forcing functions ในระบบ Multidegree of Freedom ยังจัดเป็นศาสตร์ขั้นสูง คาดว่าถ้าหากไม่สามารถจำลองแบบได้เป็นฟังก์ชันทางคณิต-

ศาสตร์แบบตรงไปตรงมา ก็อาจทดลองนำฟังก์ชันทางคณิตศาสตร์หลายรูปแบบมาจำลองให้ได้ transient response curve อันเดียว

2. ข้อมูลดิบของการตรวจวัดในภาคสนามเป็นสิ่งจำเป็น ในปัจจุบันนี้มีการตรวจวัดภาคสนามในเรื่องการสั่นสะเทือนเนื่องจากการระเบิด โดยกองการเหมืองแร่ และกองสิ่งแวดล้อม ทรัพยากรธรณี กรมทรัพยากรธรณี

คณะผู้วิจัยได้ร่วมตรวจวัดกับหน่วยงานของกรมทรัพยากรธรณีดังกล่าวในเหมืองหินปูนหลายแห่งบริเวณจังหวัดสระบุรีและลพบุรี ในการทำรายงานการวิจัยปีที่สองของโครงการพัฒนาแบบจำลองนี้ จะได้นำข้อมูลดิบจากเหมืองหินปูนดังกล่าว มาทดสอบเปรียบเทียบกับค่าที่ได้จากแบบจำลองเชิงคณิตศาสตร์

3. การป้องกันผลกระทบในเรื่องการสั่นสะเทือนจากการระเบิดเป็นสิ่งจำเป็น แต่การจะระบุกฎเกณฑ์ก่อนการศึกษาทางทฤษฎีและภาคสนามคงเป็นไปไม่ได้ จุดมุ่งหมายสำคัญอันหนึ่งคือ การพัฒนาแบบจำลองในสิ่งที่ผู้ประกอบการ ผู้ได้รับผลกระทบยอมรับได้ ซึ่งแนวทางการป้องกันนี้จะได้นำเสนอต่อไปในงานวิจัยปีต่อไป

ภาคผนวก

SOURCE CODES ของโปรแกรม VIB.EXE

ในหน้าที่ 97 ถึง 157 เป็น source codes เป็นภาษา C^{++} สำหรับโปรแกรม VIB.EXE

Source codes ที่ print มานี้ใช้นามสกุล file.CPP ทั้งหมด รายละเอียดของแต่ละโปรแกรมมีดังนี้

1. ภาคผนวก ก. เป็น File VIB.CPP เป็นโปรแกรมที่ใช้เขียนเมนูและใช้เรียกโปรแกรมคำนวณในระบบ SDF และระบบ N-DOF ได้ print เพื่อแสดง source code ของไฟล์นี้ตั้งแต่หน้าที่ 97 ถึงหน้าที่ 98

2. ภาคผนวก ข. เป็น File KUTTA.CPP เป็นโปรแกรมที่ใช้ในการคำนวณของระบบ SDF เพื่อหาค่าการเปลี่ยนตำแหน่งและความเร็วเมื่อเทียบกับเวลาได้ print เพื่อแสดง source code ของไฟล์นี้ตั้งแต่หน้าที่ 99 ถึงหน้าที่ 117

3. ภาคผนวก ค. เป็น File POLY.CPP เป็นโปรแกรมที่ใช้ในการคำนวณของระบบ N-DOF เพื่อหาค่า eigenvalues และ eigenvectors โดยวิธีทาง Gaussian elimination ได้ print เพื่อแสดง source code ของไฟล์นี้ตั้งแต่หน้าที่ 118 ถึงหน้าที่ 139

4. ภาคผนวก ง. เป็น File ITERATE.CPP เป็นโปรแกรมที่ใช้ในการหาค่า eigenvalues และ eigenvectors ในระบบ N-DOF โดยการคำนวณซ้ำตาม Power method หรือเรียกว่า Stodola-Vianello method ได้ print เพื่อแสดง source code ของไฟล์นี้ตั้งแต่หน้าที่ 140 ถึงหน้าที่ 146

5. ภาคผนวก จ. เป็น File CHOLJAC.CPP เป็นโปรแกรมที่ใช้หาค่า eigenvalues และ eigenvectors ที่มี แมตริกซ์ขนาดใหญ่ โดยใช้วิธี Choleski decomposition ร่วมกับวิธี Jacobi diagonalization ได้ print เพื่อแสดง source code ของไฟล์นี้ตั้งแต่หน้า 147 ถึง 157

ภาคผนวก ก.

FILE VIB.CPP

```
#include <alloc.h>
#include <bios.h>
#include <conio.h>
#include <dos.h>
#include <graphics.h>
#include <math.h>
#include <mem.h>
#include <process.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern void cmain();
extern int cdecl gprintf(char color, char *fmt, ... );
extern void imain();
extern void kmain();
extern void locate(int x, int y);
extern void pmain();

extern char ans,choice;


void pformat();

void main()
{
 int gdriver = DETECT, gmode, errorcode;
// char ans;

  initgraph(&gdriver, &gmode, "");
  errorcode = graphresult();
  if(errorcode != grOk)
  {
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any key to halt:");
   getch();
   exit(1);
  }
 Start:
  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(30,0);
  gprintf(LIGHTCYAN," VIBRATION METHOD ");
  locate(25,3);
  gprintf(LIGHTGREEN,"1.RUNGE-KUTTA");
  locate(25,4);
  gprintf(LIGHTGREEN,"2.POLYNOMIAL");
```

```
  locate(25,5);
  gprintf(LIGHTGREEN,"3.ITERATION");
  locate(25,6);
  gprintf(LIGHTGREEN,"4.CHOLJAC");
  locate(25,7);
  gprintf(LIGHTGREEN,"5.QUIT");
  locate(25,9);
  gprintf(YELLOW,"Enter Selection");
  ans = getche();
  locate(41,8);
  gprintf(LIGHTGREEN,"%c",ans);
  cleardevice();

  switch(ans)
  {
   case '1' : kmain();
              if( ans=='n' || ans=='N' )
             goto Start;
              break;

   case '2' : pmain();
              if( choice=='4')
             goto Start;
              break;

   case '3' : imain();
              if((choice != 'Y') || (choice != 'y'));
             goto Start;
//            break;

   case '4' : cmain();
              if( choice=='4')
             goto Start;
              break;

   case '5' : closegraph();
              exit(0);
              break;
  }

}
```

ภาคผนวก ข.

FILE KUTTA.CPP

```
/**********************/
/* Runga Kutta Method */
/**********************/
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <conio.h>
#include <dos.h>
#include <bios.h>
#include <mem.h>
#include <alloc.h>
#include <math.h>
#include <string.h>
#include <stdarg.h>

#define    kbEsc        0x011b
#define    kbBack       0x0e08
#define    kbTab        0x0f09
#define    kbEnter      0x1c0d
#define    kbHome       0x4700
#define    kbUp         0x4800
#define    kbPgUp       0x4900
#define    kbEnd        0x4f00
#define    kbDown       0x5000
#define    kbPgDn       0x5100
#define    kbIns        0x5200
#define    kbDel        0x5300
#define    kbLeft       0x4b00
#define    kbRight      0x4d00
#define    kbInsState   0x0080
#define    kbCtrlLeft   0x7300


#define false 0
#define true  1


typedef unsigned char logical;


double c,m,k,xx,yy,tt,p, h,Y[4],X[4],F[4],T[4],d;
double xlim,ylim,xval[81],yval[81],force[41],times[41];
int nwrt;
int i,j,N;
```

```
char f900[] = "t[%2d] = ";
char f910[] = "f(t[%2d]) = ";
char f920[] = "period, T= %+-4.2e maximum amplitude, A= %+-
4.2e\n";
char f930[] = "\nperiod, T= %+-4.2e maximum amplitude, A= %+-
4.2e\n";
char f940[] = "\nRunga-Kutta Program\n";
char f950[] = "        =      Problem %2d      =    \n";
char f960[] = "%+-4.2e d2x/dx2 + %+-4.2e dx/dt + %+-4.2e x = f
(t)\n";
char f970[] = "f(%+-4.2e) = %+-4.2e";
char f980[] = "x(0) = %+-4.2e";
char f990[] = "dx/dt(0) = %+-4.2e";


FILE *handle;
char ans,pause,sav;
logical wrt;
int horiz;
int dist = 0;
float f;


int evKeydown = 1;
int evNothing = 0;
unsigned int what;
unsigned int kb_code;
unsigned int kb_status;
int exitcode = kbEsc;
int confirm = 1;
int numof = 1;
int COLOR = LIGHTCYAN;
typedef struct Get
{
  int x;
  int y;
  int len;
  char *data;
  void show();
  void Default(char *);
  void init(int,int,char *,int);
  void insert(int, int);
  void replace(int, int);
  void move(int );
  void del(int );
  void cursor(int);
};

void locate(int x, int y)
{
  _AH = 2;
  _BH = 0;
  _DH = y;
  _DL = x;
```

```
  geninterrupt(0x10);
}

int cdecl gprintf(char color, char *fmt, ... )
{
va_list argptr;
char str[140];
int cnt, i;

  va_start(argptr, fmt);
  cnt = vsprintf(str, fmt, argptr);
  for (i=0; i<cnt; i++)
  {
    _AL = str[i];
    _AH = 0x0e;
    _BL = color;
    _BH = 0;
    geninterrupt(0x10);
  }
  va_end(argptr);
  return(cnt);
}

void get()
{
    kb_status = 0;
    if(bioskey(1) == 0)
    {
      what = evNothing;
      return;
    }
    what = evKeydown;
    kb_code = bioskey(0);
    kb_status = *(unsigned char *) MK_FP( 0x40, 0x17 );
    return ;
}

void Get::cursor(int c)
{
   if( c < len )
   if( ( kb_status & 0x80 ) == kbInsState )
   {
      setcolor(YELLOW);
   }
   else
      setcolor(WHITE);
   line((x+c)*8,y*16,(x+c)*8,(y*16)+15);
}

void Get::insert(int code, int c)
{
  char *m = (char*)data+c;
  if( c < len )
  {
```

```
   memmove(m+1,m,len-c-1);
  }
  *m = (char)code;
}

void Get::replace(int code, int c)
{
  *((char*)data+c) = (char) code;
}

void Get::move(int c)
{
  char *m = (char*)data+c;
  if( c < len )
  {
   memmove(m-1,m,len-c);
  }
   m = (char*)data+(len-1);
  *m = ' ';
}

void Get::del(int c)
{
  char *m = (char*)data+c;
  if( c < len )
  {
   memmove(m,m+1,len-c);
  }
   m = (char*)data+(len-1);
  *m = ' ';
}

void Get::init(int row,int col,char *d,int l)
{
  x = col;
  y = row;
  data = d;
  len = l;
}

void Get::Default(char *df)
{
  int l;
  l=strlen(df);
  if( l > len )
    memcpy(data,df,len);
  else
    memcpy(data,df,l);
}

void Get::show()
{
   int ch;
   locate(x-1,y);
```

```
      gprintf(LIGHTCYAN,"[",ch);
   locate(x+len,y);
      gprintf(LIGHTCYAN,"]",ch);
   locate(x,y);
   for(int i=0; i< len; i++ )
   {
      ch = *((char *)data+i);
      gprintf(COLOR,"%c",ch);
   }
   *((char *)data+i) = 0;

}

int entry(Get *g[])
{
   char ch;
   int c=0;
   int i;
   int ex = 0;
   for(i=0; i< numof; i++ )
   {
     g[i]->show();
   }
   kb_code = exitcode+1;
   i=0;
   g[0]->cursor(c);
   for(;ex == 0 ;)
   {
      get();
      if( what == evKeydown )
      {
      if( kb_code == exitcode )
        break;
      if( kb_code != evNothing )
      switch( kb_code )
      {
         case kbIns :
            g[i]->cursor(c);
            break;
         case kbPgUp :
            break;
         case kbPgDn :
            break;
         case kbDel :
            if( c >= 0 && c < g[i]->len)
            {
           g[i]->del(c);
           g[i]->show();
           g[i]->cursor(c);
            }
            break;
         case kbCtrlLeft :
            c=0;
            g[i]->cursor(c);
```

```
            break;
          case kbLeft :
            if( c > 0 && c <= g[i]->len)
            {
            c--;
            g[i]->show();
            g[i]->cursor(c);
            }
            break;
          case kbRight :
            if( c >= 0 && c < g[i]->len)
            {
            c++;
            g[i]->show();
            g[i]->cursor(c);
            }
            break;
          case kbUp :
            g[i]->show();
            if( i <= 0)
            {
            i = numof-1;
            }
            else
            {
            i--;
            }
            c = 0;
            g[i]->show();
            g[i]->cursor(c);
            break;
          case kbBack :
            if( c > 0 && c <= g[i]->len)
            {
            g[i]->move(c);
            g[i]->show();
            c--;
            g[i]->cursor(c);
            }
            break;
          case kbEnter :
            if(i >= numof-1)
            {
            if( confirm )
            {
               ex = 1;
               break;
            }
            }
          case kbDown :
nextline:
            g[i]->show();
            if(i >= numof-1)
            {
```

```
            i = 0;
            }
            else
            {
            i++;
            }
            c = 0;
            g[i]->cursor(c);
            break;
          default:
            ch = (char)kb_code;
            if( ch < ' ' || c < 0 || c > g[i]->len )
            break;
            if( ( kb_status & 0x80 ) == kbInsState )
            g[i]->insert(ch,c);
            else
            g[i]->replace(ch,c);
            g[i]->show();
            c++;
            if( c > g[i]->len )
            {
            c = g[i]->len;
            if( !confirm )
                goto nextline;
            }
            g[i]->cursor(c);
        }
       }
   }
   for(i=0; i< numof; i++ )
   {
      g[i]->show();
   }
   return kb_code;
}


double max(double value1, double value2)
{
   return ( (value1 > value2) ? value1 : value2);
}

double ff( double tt )
{
   double f;
   int i;
      f = 0.0;
      for( i=N; i>=2; i-- )
      {
       if( tt < times[i] )
          f = force[i-1]+(tt-times[i-1])*(force[i]-force[i-1])
            /(times[i]-times[i-1]);
      }
      if( tt < times[1] )
```

```
      f=0.0;

   return f;
}

void graph( double lim, double *val )
{
      int h;
      int i;
      d = 2.0*lim/195.0;
      setcolor( LIGHTCYAN );
      line( 50,50,  50,440 );
      line( 50,245, 600,245 );
      setcolor( RED );
      for( i=1; i<= 80; i++ )
      {
     line( 50+(i*7),50, 50+(i*7),440 );
      }
      for( i=1; i<= 80; i++ )
      {
      h = (int)245+(val[i]/d);
      setcolor( LIGHTCYAN );
      if( i == 1 )
         moveto(50+(i*7),h);
      else
      {
         if( h > 440 )
           lineto(50+(i*7),440);
         if( h < 50 )
           lineto(50+(i*7),50);
         if( h>= 50 && h <= 440 )
           lineto(50+(i*7), h);
      }
      setfillstyle(SOLID_FILL, WHITE );
      if( h>= 50 && h <= 440 )
         bar(50+(i*7)-1, h-1, 50+(i*7)+1, h+1);
      }
     return;
}

void graph2(double lim2,double *val2 )
{
      int h;
      int i;
      line( 50,50,  50,440 );
      line( 50,245, 600,245 );
      setcolor( RED );
      for( i=1; i<= 80; i++ )
      {
     line( 50+(i*7),50, 50+(i*7),440 );
      }
      d = 2.0*lim2/195.0;
      for( i=1; i<= 80; i++ )
      {
```

```
        h = (int)245+(val2[i]/d);
        setcolor( WHITE ); //LIGHTMAGENTA );
        if( i == 1 )
           moveto(50+(i*7),h);
        else
        {
           if( h > 440 )
             lineto(50+(i*7),440);
           if( h < 50 )
             lineto(50+(i*7),50);
           if( h>= 50 && h <= 440 )
             lineto(50+(i*7), h);
        }
        setfillstyle(SOLID_FILL, WHITE );
        if( h>= 50 && h <= 440 )
          bar(50+(i*7)-1, h-1, 50+(i*7)+1, h+1);
       }
      return;
}

void plot( double lim, double *val )
{
       char line[75];
       int h;
       int i;
       d = 2.0*lim/75.0;

       for( i=1; i<= 40; i++ )
       {
       memset( line, ' ', 75 );
       line[75] = 0;
       line[38] = '|';
       h = (int)38+(val[i]/d);
       if( h < 75 )
       line[h] = '*';
       fprintf(handle,"\n%.75s", line);
      }
      fprintf(handle,"\n");
      return;
}

int dataentry()
{
 int i;
 static  char tmp[16], *endptr;
 static  char dm[16];
 static  char dc[16];
 static  char dk[16];
 static  char dN[16];
 static  char dt[20][16];
 static  char df[20][16];
 static  char dx0[16];
 static  char ddt[16];
```

```
Get *g[46];

dm[15] = 0;
dm[15] = 0;
dc[15] = 0;
dk[15] = 0;
dN[15] = 0;
dx0[15] = 0;
ddt[15] = 0;

for( i=0; i<46; i++ )
{
   g[i] = (Get *)malloc(sizeof(Get));
}
for( i=0; i<20; i++ )
{
   dt[i][15] = 0;
   df[i][15] = 0;
}

locate(5,7);
gprintf(LIGHTGRAY,"m =");
locate(5,8);
gprintf(LIGHTGRAY,"c =");
locate(5,9);
gprintf(LIGHTGRAY,"k =");
locate(5,10);
gprintf(LIGHTGRAY,"N =");
g[0]->init(7,12,(char *)dm,15);
g[1]->init(8,12,(char *)dc,15);
g[2]->init(9,12,(char *)dk,15);
g[3]->init(10,12,(char *)dN,15);

locate(2,25);
gprintf(LIGHTGRAY,"    x(0)=");
locate(2,26);
gprintf(LIGHTGRAY,"dx/dt(0)=");
g[44]->init(25,12,(char *)dx0,15);
g[45]->init(26,12,(char *)ddt,15);
if( nwrt == 1 )
{
   g[0]->Default("0.200e+01");
   g[1]->Default("0.800e+01");
   g[2]->Default("0.100e+03");
   g[3]->Default("4");

   g[44]->Default("0.000e+00");
   g[45]->Default("0.000e+00");
}
for( i=0; i<20; i++)
{
  locate(30,7+i);
  gprintf(LIGHTGRAY,"t(%2d) =",i+1);
}
```

```
g[4]->init(7,40,(char *)dt[0],15);
g[5]->init(7,60,(char *)df[0],15);
g[6]->init(8,40,(char *)dt[1],15);
g[7]->init(8,60,(char *)df[1],15);
g[8]->init(9,40,(char *)dt[2],15);
g[9]->init(9,60,(char *)df[2],15);
g[10]->init(10,40,(char *)dt[3],15);
g[11]->init(10,60,(char *)df[3],15);
g[12]->init(11,40,(char *)dt[4],15);
g[13]->init(11,60,(char *)df[4],15);
g[14]->init(12,40,(char *)dt[5],15);
g[15]->init(12,60,(char *)df[5],15);
g[16]->init(13,40,(char *)dt[6],15);
g[17]->init(13,60,(char *)df[6],15);
g[18]->init(14,40,(char *)dt[7],15);
g[19]->init(14,60,(char *)df[7],15);
g[20]->init(15,40,(char *)dt[8],15);
g[21]->init(15,60,(char *)df[8],15);
g[22]->init(16,40,(char *)dt[9],15);
g[23]->init(16,60,(char *)df[9],15);
g[24]->init(17,40,(char *)dt[10],15);
g[25]->init(17,60,(char *)df[10],15);
g[26]->init(18,40,(char *)dt[11],15);
g[27]->init(18,60,(char *)df[11],15);
g[28]->init(19,40,(char *)dt[12],15);
g[29]->init(19,60,(char *)df[12],15);
g[30]->init(20,40,(char *)dt[13],15);
g[31]->init(20,60,(char *)df[13],15);
g[32]->init(21,40,(char *)dt[14],15);
g[33]->init(21,60,(char *)df[14],15);
g[34]->init(22,40,(char *)dt[15],15);
g[35]->init(22,60,(char *)df[15],15);
g[36]->init(23,40,(char *)dt[16],15);
g[37]->init(23,60,(char *)df[16],15);
g[38]->init(24,40,(char *)dt[17],15);
g[39]->init(24,60,(char *)df[17],15);
g[40]->init(25,40,(char *)dt[18],15);
g[41]->init(25,60,(char *)df[18],15);
g[42]->init(26,40,(char *)dt[19],15);
g[43]->init(26,60,(char *)df[19],15);
if( nwrt == 1 )
{
   g[4]->Default("0.000e+00");
   g[6]->Default("0.250e+00");
   g[8]->Default("0.500e+00");
   g[10]->Default("0.100e+01");
   g[12]->Default("0.0");
   g[14]->Default("0.0");
   g[16]->Default("0.0");
   g[18]->Default("0.0");
   g[20]->Default("0.0");
   g[22]->Default("0.0");
   g[24]->Default("0.0");
```

```
    g[26]->Default("0.0");
    g[28]->Default("0.0");
    g[30]->Default("0.0");
    g[32]->Default("0.0");
    g[34]->Default("0.0");
    g[36]->Default("0.0");
    g[38]->Default("0.0");
    g[40]->Default("0.0");
    g[42]->Default("0.0");

    g[5]->Default("0.000e+00");
    g[7]->Default("0.100e+01");
    g[9]->Default("0.500e+00");
    g[11]->Default("0.000e+00");
    g[13]->Default("0.0");
    g[15]->Default("0.0");
    g[17]->Default("0.0");
    g[19]->Default("0.0");
    g[21]->Default("0.0");
    g[23]->Default("0.0");
    g[25]->Default("0.0");
    g[27]->Default("0.0");
    g[29]->Default("0.0");
    g[31]->Default("0.0");
    g[33]->Default("0.0");
    g[35]->Default("0.0");
    g[37]->Default("0.0");
    g[39]->Default("0.0");
    g[41]->Default("0.0");
    g[43]->Default("0.0");
}

numof = 46;
if( entry(g) == exitcode )
    return 0;

strcpy(tmp,g[0]->data);
m = strtod(tmp,&endptr);
strcpy(tmp,g[1]->data);
c = strtod(tmp,&endptr);
strcpy(tmp,g[2]->data);
k = strtod(tmp,&endptr);
strcpy(tmp,g[3]->data);
N = atoi(tmp);

strcpy(tmp,g[44]->data);
xx = strtod(tmp,&endptr);
strcpy(tmp,g[45]->data);
yy = strtod(tmp,&endptr);

strcpy(tmp,g[4]->data);
times[1] = strtod(tmp,&endptr);
strcpy(tmp,g[6]->data);
times[2] = strtod(tmp,&endptr);
```

```
strcpy(tmp,g[8]->data);
times[3] = strtod(tmp,&endptr);
strcpy(tmp,g[10]->data);
times[4] = strtod(tmp,&endptr);
strcpy(tmp,g[12]->data);
times[5] = strtod(tmp,&endptr);
strcpy(tmp,g[14]->data);
times[6] = strtod(tmp,&endptr);
strcpy(tmp,g[16]->data);
times[7] = strtod(tmp,&endptr);
strcpy(tmp,g[18]->data);
times[8] = strtod(tmp,&endptr);
strcpy(tmp,g[20]->data);
times[9] = strtod(tmp,&endptr);
strcpy(tmp,g[22]->data);
times[10] = strtod(tmp,&endptr);
strcpy(tmp,g[24]->data);
times[11] = strtod(tmp,&endptr);
strcpy(tmp,g[26]->data);
times[12] = strtod(tmp,&endptr);
strcpy(tmp,g[28]->data);
times[13] = strtod(tmp,&endptr);
strcpy(tmp,g[30]->data);
times[14] = strtod(tmp,&endptr);
strcpy(tmp,g[32]->data);
times[15] = strtod(tmp,&endptr);
strcpy(tmp,g[34]->data);
times[16] = strtod(tmp,&endptr);
strcpy(tmp,g[36]->data);
times[17] = strtod(tmp,&endptr);
strcpy(tmp,g[38]->data);
times[18] = strtod(tmp,&endptr);
strcpy(tmp,g[40]->data);
times[19] = strtod(tmp,&endptr);
strcpy(tmp,g[42]->data);
times[20] = strtod(tmp,&endptr);

strcpy(tmp,g[5]->data);
force[1] = strtod(tmp,&endptr);
strcpy(tmp,g[7]->data);
force[2] = strtod(tmp,&endptr);
strcpy(tmp,g[9]->data);
force[3] = strtod(tmp,&endptr);
strcpy(tmp,g[11]->data);
force[4] = strtod(tmp,&endptr);
strcpy(tmp,g[13]->data);
force[5] = strtod(tmp,&endptr);
strcpy(tmp,g[15]->data);
force[6] = strtod(tmp,&endptr);
strcpy(tmp,g[17]->data);
force[7] = strtod(tmp,&endptr);
strcpy(tmp,g[19]->data);
force[8] = strtod(tmp,&endptr);
strcpy(tmp,g[21]->data);
```

```
  force[9] = strtod(tmp,&endptr);
  strcpy(tmp,g[23]->data);
  force[10] = strtod(tmp,&endptr);
  strcpy(tmp,g[25]->data);
  force[11] = strtod(tmp,&endptr);
  strcpy(tmp,g[27]->data);
  force[12] = strtod(tmp,&endptr);
  strcpy(tmp,g[29]->data);
  force[13] = strtod(tmp,&endptr);
  strcpy(tmp,g[31]->data);
  force[14] = strtod(tmp,&endptr);
  strcpy(tmp,g[33]->data);
  force[15] = strtod(tmp,&endptr);
  strcpy(tmp,g[35]->data);
  force[16] = strtod(tmp,&endptr);
  strcpy(tmp,g[37]->data);
  force[17] = strtod(tmp,&endptr);
  strcpy(tmp,g[39]->data);
  force[18] = strtod(tmp,&endptr);
  strcpy(tmp,g[41]->data);
  force[19] = strtod(tmp,&endptr);
  strcpy(tmp,g[43]->data);
  force[20] = strtod(tmp,&endptr);

  for( i=0; i<46; i++ )
  {
    free(g[i]);
  }
  return 1;
}

void kmain()
{
      int gdriver = DETECT, gmode, errorcode;
      initgraph(&gdriver, &gmode, "");
      errorcode = graphresult();
      if (errorcode != grOk)
      {
       printf("Graphics error: %s\n", grapherrormsg(errorcode));
       printf("Press any key to halt:");
       getch();
       exit(1);
      }
      _setcursortype(_NORMALCURSOR);
      rectangle(0,7,getmaxx(),getmaxy()-15);
      nwrt = 1;
      wrt = false;
      locate(27,0);
      gprintf(LIGHTCYAN,"+++ RUNGE-KUTTA METHOD +++");
      locate(2,1);
      gprintf(WHITE,"This program will compute the response of a
single degree of freedom system.");
      locate(2,2);
```

```
      gprintf(WHITE,"The output can be either numerical or as a
rough plot.");

      if( nwrt == 1 )
      {
       locate(1,28);
       gprintf(YELLOW,"Do you want to save the output to
runga.dat? ");
       sav = getch();
       locate(46,28);
       gprintf(WHITE,"%c",sav);

       if( sav == 'y' || sav == 'Y' )
       {
          wrt = true;
          handle = fopen("runga.dat","w+t");
          fprintf(handle,f940);
       }
      }
Line10:
      locate(5,3);
      gprintf(LIGHTGREEN,"Equation form : m(d2x/dt2)+c
(dx/dt)+kx=f(t)");
      locate(5,4);
      gprintf(LIGHTGREEN,"f(t) is entered by inputing N points
describing the function with linear");
      locate(5,5);
      gprintf(LIGHTGREEN,"interpolation between points.  A
maximum of 20 points are allowed.");

      if( !dataentry() )
     goto Quit;
Line20:
      if( wrt )
      {
     fprintf(handle,f950,nwrt);
      }
      if( wrt )
      {
       fprintf(handle,f960,m,c,k);
       for( i=1; i<=N; i++ )
       {
          fprintf(handle,f970,times[i],force[i]);
       }
       fprintf(handle,f980, xx);
       fprintf(handle,f990, yy);
      }


      tt=0.0;
      p=1.0/sqrt(k/m);
      h=p/2.0;
      xlim=0.0;
      ylim=0.0;
```

```
for(i=1; i<=40; i++)
{
xval[i]=xx;
yval[i]=yy;
T[1]=tt;
Y[1]=yy;
X[1]=xx;
F[1]=(ff(T[1])-c*Y[1]-k*X[1])/m;
for(j=2; j<=3; j++)
{
   T[j]=tt+h/2.0;
   Y[j]=yy+h/2.0*F[j-1];
   X[j]=xx+h/2.0*Y[j-1];
   F[j]=(ff(T[j])-c*Y[j]-k*X[j])/m;
}
T[4]=tt+h;
Y[4]=yy+h*F[3];
X[4]=xx+h*Y[3];
F[4]=(ff(T[4])-c*Y[4]-k*X[4])/m;
xx=xx+h/6.0*(Y[1]+2.0*Y[2]+2.0*Y[3]+Y[4]);
yy=yy+h/6.0*(F[1]+2.0*F[2]+2.0*F[3]+F[4]);
tt=h+tt;
xlim=max(fabs(xx),xlim);
ylim=max(fabs(yy),ylim);
}


setfillstyle(SOLID_FILL, WHITE );
locate(1,28);
gprintf(YELLOW,"Do you want numerical output?(y/n)
");
ans = getch();
locate(36,28);
gprintf(WHITE,"%c",ans);


if( ans == 'y' || ans == 'Y' )
{
setfillstyle(SOLID_FILL, BLACK );
bar(1,50,638,448);
locate(1,4);
gprintf(WHITE,"        time      disp    veloc");
if( wrt )
{
   fprintf(handle,"\n    time      disp    veloc");
}
if( ans == 'y' || ans == 'Y' )
f = 0.0;
for(i=1; i<=80; i++)
{
      f+=0.5;
      if( i== 20 )
```

```
            {
            locate(40,4);
            gprintf(WHITE,"    time     disp    veloc");
            }
            if( i>=1 && i <= 20 )
            locate(5,i+5);
            if( i>20 && i <=40 )
            locate(40,i+5-20);
            if( i>40 && i <=60 )
            locate(5,i+5-40);
            if( i>60 && i <=80 )
            locate(40,i+5-60);
            gprintf(GREEN,"%2d",i);
            gprintf(LIGHTGRAY," %+-4.2e %+-4.2e %+-4.2e",h*
(double)(i-1),xval[i],yval[i]);
            if( i == 40 )
            {
            locate(1,28);
            gprintf(YELLOW,"
");
            locate(1,28);
            gprintf(YELLOW,"any key to continue...");
            getch();
            }
            if( wrt )
            fprintf(handle,"\n  %+-4.2e %+-4.2e %+-4.2e",h*
(double)(i-1),xval[i],yval[i]);
      }
      }
      locate(1,28);
      gprintf(YELLOW,"
");
      locate(1,28);
      gprintf(YELLOW,"Do you want a plot of displacements vs
time?(y/n)     ");
      ans = getch();
      locate(51,28);
      gprintf(WHITE,"%c",ans);
      dist = 0;
      if( ans == 'y' || ans == 'Y' )
      {
      line( 50,50,  50,440 );
      line( 50,245, 600,245 );


      setfillstyle(SOLID_FILL, BLACK );
      bar(1,50,638,448);


      if( wrt )
      {
         fprintf(handle,f930,p*2.*3.14159,xlim);
         fprintf(handle,"Displacements vs time :\n");
      }
```

```
        if( wrt )
          plot(xlim,xval);
        graph(xlim,xval);
        locate(1,4);
        gprintf(LIGHTCYAN,"(Displacements)");
        locate(70,16);
        gprintf(WHITE,"(times)");
        locate(20,26);
        gprintf(LIGHTCYAN,f920,p*2.*3.14159,xlim);
        dist = 1;
        }
        else
        {
        setfillstyle(SOLID_FILL, BLACK );
        bar(1,50,638,448);
        }
        locate(1,28);
        gprintf(YELLOW,"Do you want a plot of velocities vs time?
(y/n)          " );
        ans = getch();
        locate(48,28);
        gprintf(WHITE,"%c",ans);
        if( ans == 'y' || ans == 'Y' )
        {
        if( wrt )
        {
           fprintf(handle,f930, p*2.*3.14159,ylim);
           fprintf(handle,"Velocities vs time :");
        }
        if( wrt )
          plot(ylim,yval);
        graph2(ylim,yval);
        if( dist )
        {
          locate(1,4);
          gprintf(LIGHTCYAN,"(Displacements)");
          locate(70,16);
          gprintf(WHITE,"(times)");
          locate(20,26);
          gprintf(LIGHTCYAN,f920,p*2.*3.14159,xlim);
        }


        locate(1,5);
        gprintf(WHITE,"(Velocities)");
        locate(70,16);
        gprintf(WHITE,"(times)");
        locate(20,27);
        gprintf(WHITE,f920,p*2.*3.14159,ylim);
        }


Quit:
```

```
        locate(1,28);
        gprintf(YELLOW,"Another problem?(y/n)                    ");
        ans = getch();
        locate(23,28);
        gprintf(WHITE,"%c",ans);
        nwrt++;
        if( ans=='y' || ans=='Y' )
        {
          setfillstyle(SOLID_FILL, BLACK );
          bar(1,50,638,448+14);
          goto Line10;
        }
        if( wrt )
       fclose(handle);
//       closegraph();
}
```

ภาคผนวก ค.

FILE POLY.CPP

```
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>


extern int i,j;
extern char ans;


extern int cdecl gprintf(char color, char *fmt, ... );
extern void locate(int x, int y);



int  n,mode;
char  choice;
float g[20][20],s[20][20],a[20][20],qit[20][20],qi[20][20],w[20]
[20];
float temp[20][20],q[20][20],x[20][20],e[20],co[21];


void choleski(int n, float w[20][20], float qit[20][20], float qi
[20][20]);

void coeff(int n, float a[20][20], float pcf[21]);

void gauss(int n, int mode, float a[20][20], float e[20], float x
[20][20]);

void matrix(int col, int row, char *ch, float t[20][20], char
*head);

void mproduct(int n, float u[20][20], float w[20][20], float x
[20][20]);

void prnt_gauss();

void prnt_matrix(int col, int row, char *ch, float x[20][20],
char *head);

void prnt_roots(int col, int row, char *head, char *detail);
```

```
void roots(int n, float co[21], float r[20]);

void pmain()
{
Pstart:
  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(34,0);
  gprintf(LIGHTCYAN," POLYNOMIAL ");
  locate(2,1);
  gprintf(WHITE,"This program will determine the coefficients of
the polynomial given by");
  locate(2,2);
  gprintf(WHITE,"det|[M]-lambda[K]|.  If these coefficients are
known,  the roots to the");
  locate(2,3);
  gprintf(WHITE,"polynomial can be calculated. The roots must be
real.If [M] and [K] are");
  locate(2,4);
  gprintf(WHITE,"input then the eigenvalues and eigenvectors can
be determined.");
  locate(2,5);
  gprintf(WHITE,"The maximum order of the problem is 20.
");
  locate(15,10);
  gprintf(LIGHTGREEN,"     (1) Coefficients of det|-X[M]+[K]|
");
  locate(15,11);
  gprintf(LIGHTGREEN,"         (Input [M] and [K])
");
  locate(15,12);
  gprintf(LIGHTGREEN,"     (2) Roots to polynomial (Eigenvalues)
");
  locate(15,13);
  gprintf(LIGHTGREEN,"         (Input coefficients)
");
  locate(15,14);
  gprintf(LIGHTGREEN,"     (3) Eigenvalues for -lambda [M]+[K]
");
  locate(15,15);
  gprintf(LIGHTGREEN,"         (Input [M], [K], and lambda)
");
  locate(15,16);
  gprintf(LIGHTGREEN,"     (4) Quit
");
  locate(15,18);
  gprintf(YELLOW,"  Enter selection (1-4):  ");
  choice = getch();
  locate(41,8);
  gprintf(LIGHTGREEN,"%c",choice);
  cleardevice();
```

```
  switch(choice)
  {
   case '1' :
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(24,0);
        gprintf(LIGHTCYAN," COEFFICIENTS OF DET[-X[M]+[K]] ");
        locate(5,1);
        gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
        cin >> n;
        matrix(24,0,"M",g," COEFFICIENTS OF DET[-X[M]+[K]] ");
        matrix(24,0,"K",s," COEFFICIENTS OF DET[-X[M]+[K]] ");
        choleski(n, s, qit, qi);
        mproduct(n, qi, qit, temp);
        mproduct(n, temp, g, a);
        coeff(n, a, co);
        cleardevice();
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(24,0);
        gprintf(LIGHTCYAN," COEFFICIENTS OF DET[-X[M]+[K]] ");
        locate(5,1);
        gprintf(LIGHTGREEN,"The coefficients of");
        locate(7,3);
        gprintf(LIGHTGREEN,"C(N)X^N + C(N-1)X^(N-1) + ... + C(1)X
+ C(0) = 0  are :");


        for( i=0; i<=n; i++)
        {
      locate(10,i+5);
      gprintf(LIGHTGRAY,"C[%d] = ",i);
      locate(18,i+5);
      gprintf(LIGHTCYAN,"%.4e",co[i]);
      printf("\n");
        }


        locate(1,28);
        gprintf(YELLOW,"Determine the roots to this polynomial
(Y/N)? ");
        ans = getch();
        locate(47,28);
        gprintf(WHITE,"%c",ans);


        if ((ans == 'Y') || (ans == 'y'))
        {
      roots(n, co, e);
      prnt_roots(23,0," ROOTS TO POLYNOMIAL (EIGENVALUES) ",
              "The roots (eigenvalues) are : ");


      locate(1,27);
```

```
      gprintf(YELLOW,"Determine the eigenvectors (Y/N)? ");
      ans = getch();
      locate(35,27);
      gprintf(WHITE,"%c",ans);


      if ((ans == 'Y') || (ans == 'y'))
      {
       gauss(n, mode, a, e, x);
       prnt_gauss();
      }
      goto Pstart;


       }
       else
      goto Pstart;


//       break;


   case '2':
       _setcursortype(_NORMALCURSOR);
       rectangle(0,7,getmaxx(),getmaxy()-15);
       locate(23,0);
       gprintf(LIGHTCYAN," ROOTS TO POLYNOMIAL (EIGENVALUES) ");
       locate(5,1);
       gprintf(WHITE,"Enter Polynomial order : ");
       cin >> n;
       do
       {


      cleardevice();
      _setcursortype(_NORMALCURSOR);
      rectangle(0,7,getmaxx(),getmaxy()-15);
      locate(23,0);
      gprintf(LIGHTCYAN," ROOTS TO POLYNOMIAL (EIGENVALUES) ");
      locate(5,1);
      gprintf(WHITE,"Enter Polynomial order : ");
      locate(30,1);
      gprintf(LIGHTGRAY,"%d",n);
      locate(5,2);
      gprintf(LIGHTGREEN,"The polynomial is of the form : ");
      locate(7,3);
      gprintf(LIGHTGREEN,"C(N)X^N + C(N-1)X^(N-1) + ... + C(1)X +
C(0) = 0 are");
      locate(5,4);
      gprintf(LIGHTGREEN,"Enter the  Coefficients, C(i)");
      for( i=0; i<=n; i++)
      {
       locate(10,i+5);
       gprintf(LIGHTCYAN,"C[%d] = ",i);
```

```
       cin >> co[i];
      }
      locate(1,28);
      gprintf(YELLOW,"Are these correct?(Y/N)");
      ans = getch();
       }
       while ((ans == 'N') || (ans == 'n'));
       roots(n, co, e);
       prnt_roots(23,0," ROOTS TO POLYNOMIAL (EIGENVALUES) ",
            "The roots (eigenvalues) are : ");
       getch();
       goto Pstart;
//       break;


   case '3':
       _setcursortype(_NORMALCURSOR);
       rectangle(0,7,getmaxx(),getmaxy()-15);
       locate(24,0);
       gprintf(LIGHTCYAN," EIGENVALUES FOR -LAMBDA [M]+[K] ");
       locate(5,1);
       gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
       cin >> n;
       matrix(24,0,"M",g," EIGENVALUES FOR -LAMBDA [M]+[K] ");
       matrix(24,0,"K",s," EIGENVALUES FOR -LAMBDA [M]+[K] ");
       choleski(n, s, qit, qi);
       mproduct(n, qi, qit, temp);
       mproduct(n, temp, g, a);


       do
       {
      cleardevice();
      _setcursortype(_NORMALCURSOR);
      rectangle(0,7,getmaxx(),getmaxy()-15);
      locate(24,0);
      gprintf(LIGHTCYAN," EIGENVALUES FOR -LAMBDA [M]+[K] ");
      mode = 0;
      locate(5,1);
      gprintf(WHITE,"Enter Eigenvalue : ");
      cin >> e[0];


      gauss(n, mode, a, e, x);


      prnt_gauss();


      locate(1,28);
      gprintf(YELLOW,"Another mode (Y/N)?
");
      ans = getch();
      locate(21,28);
```

```
      gprintf(WHITE,"%c",ans);
        }
        while ((ans == 'Y') || (ans == 'y'));
        goto Pstart;
//        break;


   case '4':
       break;
  }
}


void choleski(int n, float w[20][20], float qit[20][20], float qi
[20][20])
/*  DECOMP OF MATRIX W SO THAT [QT]*[Q]=[W]
    RETURN [QI](INVERSE OF [QI]) AND [QIT]
    TRANSPOSE OF [QI]                        */
{
 int i,j,k,m;
 int im1,ip1;
 float sum;


 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   q[i][j] = 0;
   qi[i][j] = 0;
  }
 }


 q[0][0] = sqrt(w[0][0]);


 for( j=1; j<n; j++)
 {
  q[0][j] = w[0][j]/q[0][0];
 }


 for( i=1; i<n; i++)
 {
  im1 = i-1;
  sum = 0;
  for( k=0; k<=im1; k++)
  {
   sum = sum+pow(q[k][i],2);
  }
  q[i][i] = sqrt(w[i][i]-sum);
  ip1 = i+1;
  for( j=ip1; j<n; j++)
```

```
  {
   sum = 0;
   for( k=0; k<=im1; k++)
   {
    sum = sum+q[k][i]*q[k][j];
   }
   q[i][j] = 1.*(w[i][j]-sum)/q[i][i];
  }
 }

/* COMPUTE INVERSE OF [Q]    */


 for( i=0; i<n; i++)
 {
  qi[i][i] = 1./q[i][i];
 }


 for( m=1; m<n; m++)
 {
  for( j=m; j<n; j++)
  {
   i = j-m;
   ip1 = i+1;
   sum = 0;
   for( k=ip1; k<=j; k++)
   {
    sum = sum+q[i][k]*qi[k][j];
   }
   qi[i][j] = -sum/q[i][i];
  }
 }

 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   qit[i][j] = qi[j][i];
  }
 }

}


void coeff(int n, float a[20][20], float pcf[21])


/*   RETURNS COEFFICIENTS OF POLYNOMIAL GIVEN
     BY DET[A]                                    */


{
 int i,j,k;
```

```
int km;
float b[20][20], c[20][20];
float s[20], p[20];

for( i=0; i<n; i++)
{
 p[i] = 0;
 for( j=0; j<n; j++)
 {
  b[i][j] = 0;
 }
 b[i][i] = 1;
}

for( k=0; k<n; k++)
{
 s[k] = 0;
 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   c[i][j] = b[i][j];
  }
 }
 mproduct(n, c, a, b);


 for( j=0; j<n; j++)
 {
  s[k] = s[k]+b[j][j];
 }


 p[0] = -s[0];
 if( k != 0)
 {
  km = k-1;
  for( i=0; i<=km; i++)
  {
   p[k] = p[k]-(1.0/(k+1))*(s[i]*p[k-1-i]);
  }
  p[k] = p[k]-s[k]/(k+1);
 }
}

for( i=0; i<n; i++)
{
 pcf[i] = p[n-1-i];
}

pcf[n] = 1;


}
```

```
void gauss(int n, int mode, float a[20][20], float e[20], float x
[20][20])


/*   RETURNS EIGENVECTORS OF [A] GIVEN [A]
     [A] AND EIGENVECTORS [E]                          */


{
 int i, j, k, l;
 float w[20][20];
/* float e[20];*/
 float sum;


 for( l=0; l<=mode; l++)
 {
  for( i=0; i<n; i++)
  {
   for( j=0; j<n; j++)
   {
    w[i][j] = a[i][j];
   }
   w[i][i] = w[i][i] - e[l];
  }
  for( i=0; i<n-2; i++)
  {
   for( j=i+1; j<n; j++)
   {
    for( k=i+1; k<n; k++)
    {
     w[j][k] = w[j][k]-w[i][k]*w[j][i]/w[i][i];
    }
   }
  }
  x[n-1][l] = 1;
  for( i=n-2; i>=0; i--)
  {
   sum = 0;
   for( j=i+1; j<n; j++)
   {
    sum = sum+w[i][j]*x[j][l];
   }
   x[i][l] = -sum/w[i][i];
  }


 }
}
```

```
void matrix(int col, int row, char *ch, float t[20][20], char
*head)
{
 do
 {
  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(col,row);
  gprintf(LIGHTCYAN,"%s",head);
  locate(5,1);
  gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
  locate(40,1);
  gprintf(LIGHTGRAY,"%d",n);
  locate(5,2);
  gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
  mode = n-1;


  if (n <= 5)
  {
   for( i=0; i<n; i++)
    for( j=0; j<n; j++)
    {
     locate(5+(15*j),3+i);
     gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
     cin >> t[i][j];
    }
  }
  else
  {
   if (n<=10)
   {
    for( i=0; i<n; i++)
     for( j=0; j<5; j++)
     {
      locate(5+(15*j),3+i);
      gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
      cin >> t[i][j];
     }
    locate(1,28);
    gprintf(YELLOW,"Press Any Key to Continue ");
    ans = getch();
    cleardevice();
    _setcursortype(_NORMALCURSOR);
    rectangle(0,7,getmaxx(),getmaxy()-15);
    locate(col,row);
    gprintf(LIGHTCYAN,"%s",head);
    locate(5,1);
    gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
    locate(40,1);
    gprintf(LIGHTGRAY,"%d",n);
    locate(5,2);
    gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
```

```
 for( i=0; i<n; i++)
  for( j=5; j<n; j++)
  {
   locate(5+(15*(j-5)),3+i);
   gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
   cin >> t[i][j];
  }
}
else
{
 if (n<=15)
 {
  for( i=0; i<n; i++)
   for( j=0; j<5; j++)
   {
    locate(5+(15*j),3+i);
    gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
    cin >> t[i][j];
   }
  locate(1,28);
  gprintf(YELLOW,"Press Any Key to Continue ");
  ans = getch();
  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(col,row);
  gprintf(LIGHTCYAN,"%s",head);
  locate(5,1);
  gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
  locate(40,1);
  gprintf(LIGHTGRAY,"%d",n);
  locate(5,2);
  gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
  for( i=0; i<n; i++)
   for( j=5; j<10; j++)
   {
    locate(5+(15*(j-5)),3+i);
    gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
    cin >> t[i][j];
   }
  locate(1,28);
  gprintf(YELLOW,"Press Any Key to Continue ");
  ans = getch();
  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(col,row);
  gprintf(LIGHTCYAN,"%s",head);
  locate(5,1);
  gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
  locate(40,1);
  gprintf(LIGHTGRAY,"%d",n);
  locate(5,2);
  gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
```

```
 for( i=0; i<n; i++)
  for( j=10; j<n; j++)
  {
   locate(5+(15*(j-10)),3+i);
   gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
   cin >> t[i][j];
  }
}
else
{
 for( i=0; i<n; i++)
  for( j=0; j<5; j++)
  {
   locate(5+(15*j),3+i);
   gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
   cin >> t[i][j];
  }
 locate(1,28);
 gprintf(YELLOW,"Press Any Key to Continue ");
 ans = getch();
 cleardevice();
 _setcursortype(_NORMALCURSOR);
 rectangle(0,7,getmaxx(),getmaxy()-15);
 locate(col,row);
 gprintf(LIGHTCYAN,"%s",head);
 locate(5,1);
 gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
 locate(40,1);
 gprintf(LIGHTGRAY,"%d",n);
 locate(5,2);
 gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
 for( i=0; i<n; i++)
  for( j=5; j<10; j++)
  {
   locate(5+(15*(j-5)),3+i);
   gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
   cin >> t[i][j];
  }
 locate(1,28);
 gprintf(YELLOW,"Press Any Key to Continue ");
 ans = getch();
 cleardevice();
 _setcursortype(_NORMALCURSOR);
 rectangle(0,7,getmaxx(),getmaxy()-15);
 locate(col,row);
 gprintf(LIGHTCYAN,"%s",head);
 locate(5,1);
 gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
 locate(40,1);
 gprintf(LIGHTGRAY,"%d",n);
 locate(5,2);
 gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
 for( i=0; i<n; i++)
  for( j=10; j<15; j++)
```

```
      {
       locate(5+(15*(j-10)),3+i);
       gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
       cin >> t[i][j];
      }
     locate(1,28);
     gprintf(YELLOW,"Press Any Key to Continue ");
     ans = getch();
     cleardevice();
     _setcursortype(_NORMALCURSOR);
     rectangle(0,7,getmaxx(),getmaxy()-15);
     locate(col,row);
     gprintf(LIGHTCYAN,"%s",head);
     locate(5,1);
     gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
     locate(40,1);
     gprintf(LIGHTGRAY,"%d",n);
     locate(5,2);
     gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
     for( i=0; i<n; i++)
      for( j=15; j<n; j++)
      {
       locate(5+(15*(j-15)),3+i);
       gprintf(LIGHTCYAN,"%c[%d][%d] = ",*ch,i+1,j+1);
       cin >> t[i][j];
      }
    }
   }
  }


  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(col,row);
  gprintf(LIGHTCYAN,"%s",head);
  locate(5,1);
  gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
  locate(40,1);
  gprintf(LIGHTGRAY,"%d",n);
  locate(5,2);
  gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
  prnt_matrix(24,0,ch,t," COEFFICIENTS OF DET[-X[M]+[K]] ");
  locate(1,28);
  gprintf(YELLOW,"Is this correct?(Y/N) ");
  ans = getch();
  locate(23,28);
  gprintf(WHITE,"%c",ans);
 }
 while ((ans == 'N') || (ans == 'n'));
}
```

```
void mproduct(int n, float u[20][20], float w[20][20], float x
[20][20])
/*  RETURNS X, WHERE X=U*W  */
{
 int i,j,k;
/* float u[20][20], w[20][20], x[20][20];
  */
 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   x[i][j] = 0;
   for( k=0; k<n; k++)
   {
     x[i][j] = x[i][j]+u[i][k]*w[k][j];
   }
  }
 }

}


void prnt_gauss()
{
 int l;
 float den,sum;


 locate(1,28);
 gprintf(YELLOW,"Do you want orthonormal eigenvectors? (Y/N) ");
 ans = getch();
 locate(45,28);
 gprintf(WHITE,"%c",ans);


 if ((ans == 'Y') || (ans == 'y'))
 {
  for( j=0; j<=mode; j++)
  {
   sum = 0;
   for( i=0; i<n; i++)
   {
    sum = sum+x[i][j]*x[i][j];
   }
   den = sqrt(sum);
   for( i=0; i<n; i++)
   {
    x[i][j] = x[i][j]/den;
   }
  }
 }
```

```
 if ( choice == '3')
 {
  locate(7,3);
  gprintf(LIGHTGREEN,"The eigenvector is: ");
  for( i=0; i<n; i++)
  {
   locate(10,5+i);
   gprintf(LIGHTCYAN,"%.4e",x[i][0]);
  }
 }
 else
 {
  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(24,0);
  gprintf(LIGHTCYAN," EIGENVALUES FOR -LAMBDA [M]+[K] ");
  locate(5,1);
  gprintf(LIGHTGREEN,"The eigenvectors are: ");
  prnt_matrix(24,0,"",x," EIGENVALUES FOR -LAMBDA [M]+[K] ");
  getch();
/*  for( l=0; l<n; l++)
  {
   printf("\n\n  The eigenvalue   is: %.4e\n ",e[l]);
   printf("\n  The eigenvectors is: %.4e\n",x[0][l]);
   for( i=1; i<n; i++)
   {
    printf("                        %.4e\n",x[i][l]);
   }
   printf("\n  Please Press Any Key to continue ");
   ans = getch();
  }*/
 }

}



void prnt_matrix(int col, int row, char *ch, float x[20][20],
char *head)
{
 if (n <= 5)
 {
  for( i=0; i<n; i++)
   for( j=0; j<n; j++)
   {
    locate(7+(13*j),3+i);
    gprintf(LIGHTCYAN,"%.4e",x[i][j]);
   }
 }
 else
 {
```

```
if (n<=10)
{
 for( i=0; i<n; i++)
  for( j=0; j<5; j++)
  {
   locate(7+(13*j),3+i);
   gprintf(LIGHTCYAN,"%.4e",x[i][j]);
  }
 locate(1,28);
 gprintf(YELLOW,"Press Any Key to Continue ");
 getch();
 cleardevice();
 _setcursortype(_NORMALCURSOR);
 rectangle(0,7,getmaxx(),getmaxy()-15);
 locate(col,row);
 gprintf(LIGHTCYAN,"%s",head);
 if(ch != NULL)
 {
  locate(5,1);
  gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
  locate(40,1);
  gprintf(LIGHTGRAY,"%d",n);
  locate(5,2);
  gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
 }
 for( i=0; i<n; i++)
  for( j=5; j<n; j++)
  {
   locate(7+(13*(j-5)),3+i);
   gprintf(LIGHTCYAN,"%.4e",x[i][j]);
  }
}
else
{
 if (n<=15)
 {
  for( i=0; i<n; i++)
   for( j=0; j<5; j++)
   {
    locate(7+(13*j),3+i);
    gprintf(LIGHTCYAN,"%.4e",x[i][j]);
   }
  locate(1,28);
  gprintf(YELLOW,"Press Any Key to Continue ");
  getch();
  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(col,row);
  gprintf(LIGHTCYAN,"%s",head);
```

```
 if(ch != NULL)
 {
  locate(5,1);
  gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
  locate(40,1);
  gprintf(LIGHTGRAY,"%d",n);
  locate(5,2);
  gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
 }
 for( i=0; i<n; i++)
  for( j=5; j<10; j++)
  {
   locate(7+(13*(j-5)),3+i);
   gprintf(LIGHTCYAN,"%.4e",x[i][j]);
  }
 locate(1,28);
 gprintf(YELLOW,"Press Any Key to Continue ");
 getch();
 cleardevice();
 _setcursortype(_NORMALCURSOR);
 rectangle(0,7,getmaxx(),getmaxy()-15);
 locate(col,row);
 gprintf(LIGHTCYAN,"%s",head);
 if(ch != NULL)
 {
  locate(5,1);
  gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
  locate(40,1);
  gprintf(LIGHTGRAY,"%d",n);
  locate(5,2);
  gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
 }
 for( i=0; i<n; i++)
  for( j=10; j<n; j++)
  {
   locate(7+(13*(j-10)),3+i);
   gprintf(LIGHTCYAN,"%.4e",x[i][j]);
  }
}
else
{
 for( i=0; i<n; i++)
  for( j=0; j<5; j++)
  {
   locate(7+(13*j),3+i);
   gprintf(LIGHTCYAN,"%.4e",x[i][j]);
  }
 locate(1,28);
 gprintf(YELLOW,"Press Any Key to Continue ");
 getch();
 cleardevice();
 _setcursortype(_NORMALCURSOR);
 rectangle(0,7,getmaxx(),getmaxy()-15);
 locate(col,row);
```

```
gprintf(LIGHTCYAN,"%s",head);
if(ch != NULL)
{
 locate(5,1);
 gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
 locate(40,1);
 gprintf(LIGHTGRAY,"%d",n);
 locate(5,2);
 gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
}
for( i=0; i<n; i++)
 for( j=5; j<10; j++)
 {
  locate(7+(13*(j-5)),3+i);
  gprintf(LIGHTCYAN,"%.4e",x[i][j]);
 }
locate(1,28);
gprintf(YELLOW,"Press Any Key to Continue ");
getch();
cleardevice();
_setcursortype(_NORMALCURSOR);
rectangle(0,7,getmaxx(),getmaxy()-15);
locate(col,row);
gprintf(LIGHTCYAN,"%s",head);
if(ch != NULL)
{
 locate(5,1);
 gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
 locate(40,1);
 gprintf(LIGHTGRAY,"%d",n);
 locate(5,2);
 gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
}
for( i=0; i<n; i++)
 for( j=10; j<15; j++)
 {
  locate(7+(13*(j-10)),3+i);
  gprintf(LIGHTCYAN,"%.4e",x[i][j]);
 }
locate(1,28);
gprintf(YELLOW,"Press Any Key to Continue ");
getch();
cleardevice();
_setcursortype(_NORMALCURSOR);
rectangle(0,7,getmaxx(),getmaxy()-15);
locate(col,row);
gprintf(LIGHTCYAN,"%s",head);
if(ch != NULL)
{
 locate(5,1);
 gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
 locate(40,1);
 gprintf(LIGHTGRAY,"%d",n);
 locate(5,2);
```

```
     gprintf(LIGHTGREEN,"Enter Matrix %c :",*ch);
    }
    for( i=0; i<n; i++)

     for( j=15; j<n; j++)
     {
      locate(7+(13*(j-15)),3+i);
      gprintf(LIGHTCYAN,"%.4e",x[i][j]);
     }
   }
  }
 }
}



void prnt_roots(int col, int row, char *head, char *detail)
{
 cleardevice();
 _setcursortype(_NORMALCURSOR);
 rectangle(0,7,getmaxx(),getmaxy()-15);
 locate(col,row);
 gprintf(LIGHTCYAN,"%s",head);
 locate(5,1);
 gprintf(LIGHTGREEN,"%s",detail);

 if (n<=5)
 {
  for( i=0; i<n; i++)
  {
   locate(7+(13*i),3);
   gprintf(LIGHTCYAN,"%.4e",e[i]);
  }
 }
 else
 {
  if (n<=10)
  {
   for( i=0; i<5; i++)
   {
    locate(7+(13*i),3);
    gprintf(LIGHTCYAN,"%.4e",e[i]);
   }
   for( i=5; i<n; i++)
   {
    locate(7+(13*i),4);
    gprintf(LIGHTCYAN,"%.4e",e[i]);
   }
  }
  else
  {
   if (n<=15)
   {
    for( i=0; i<5; i++)
```

```
    {
     locate(7+(13*i),3);
     gprintf(LIGHTCYAN,"%.4e",e[i]);
    }
    for( i=5; i<10; i++)
    {
     locate(7+(13*i),4);
     gprintf(LIGHTCYAN,"%.4e",e[i]);
    }
    for( i=10; i<n; i++)
    {
     locate(7+(13*i),5);
     gprintf(LIGHTCYAN,"%.4e",e[i]);
    }
   }
   else
   {
    for( i=0; i<5; i++)
    {
     locate(7+(13*i),3);
     gprintf(LIGHTCYAN,"%.4e",e[i]);
    }
    for( i=5; i<10; i++)
    {
     locate(7+(13*i),4);
     gprintf(LIGHTCYAN,"%.4e",e[i]);
    }
    for( i=10; i<15; i++)
    {
     locate(7+(13*i),5);
     gprintf(LIGHTCYAN,"%.4e",e[i]);
    }
    for( i=15; i<n; i++)
    {
     locate(7+(13*i),6);
     gprintf(LIGHTCYAN,"%.4e",e[i]);
    }
   }
  }
 }
}


void roots(int n, float co[21], float r[20])
/*    RETURNS ROOTS OF N ORDER POLYNOMIAL
      GIVEN N+1 COEFFICIENTS C                */
{
 int i,j,k,l,m;
 float f[2];
 float ff, fprime, fsum, del,x;
 float delta, emax;
```

```
emax = 1e-6;
fsum = fabs(co[n-1]/co[n]);
del =  fsum/100;
l = 0;
f[0] = co[0];
f[1] = 0;


do
{
 for( j=1; j<=99; j++)
 {
  x = del*j;


  for( k=0; k<=n; k++)
  {
   m = k;
   f[1] = f[1]+co[k]*pow(x,m);
  }


  if( f[1] != 0)
  {
   if( f[1]*f[0] < 0)
   {
    r[l] = x;
    l = l+1;
   }
  }


  f[0] = f[1];
  f[1] = 0;
 }
}
while (l != n);


for( i=0; i<l; i++)
{
 x = r[i];
 delta = 1;


 while (fabs(delta) > emax)
 {
  ff = 0;
  fprime = 0;
```

```
    for( k=0; k<=n; k++)
    {
     m = k;
     ff = ff+co[k]*pow(x,m);
     fprime = fprime+m*co[k]*pow(x,(m-1));
    }
    delta = ff/fprime;
    r[i] = x-delta;
    x = r[i];
   }

 }

}
```

ภาคผนวก ง.

FILE ITERATE.CPP

```
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>


#define false 0
#define true 1


typedef unsigned char logical;


extern int cdecl gprintf(char color, char *fmt, ... );
extern void locate(int x, int y);
extern void matrix(int col, int row, char *ch, float t[20][20],
char *head);


extern int i,j,n,mode;
extern char ans,choice;
extern float g[20][20],s[20][20],a[20][20],qit[20][20],qi[20]
[20];
extern float q[20][20],d[20][20],x[20],p[20],w[20][20];


int  iter,itmax;
float ev,oev,den,sum,tol;
float qitg[20][20],ad[20][20];
logical on;


void icholeski(int n, float w[20][20], float q[20][20], float qi
[20][20]);
void deflate(int n, float g[20][20], float p[20], float d[20]
[20]);
void improduct(int n, float u[20][20], float w[20][20], float z
[20][20]);
void vproduct(int n, float ad[20][20], float x[20], float p[20]);

void imain()
{
 tol = 1e-10;
 itmax = 50;
```

```
 do
 {
  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(35,0);
  gprintf(LIGHTCYAN," ITERATION ");
  locate(2,1);
  gprintf(WHITE,"This program will determine eigenvalues and
eigenvectors using matrix  ");
  locate(2,2);
  gprintf(WHITE,"iteration. Input matrix order N (maximum of 20),
and matrices [M] and  ");
  locate(2,3);
  gprintf(WHITE,"[K].");


  on = false;


  locate(1,28);
  gprintf(YELLOW,"Do you want orthonormal eigenvectors? (Y/N) ");
  ans = getch();
  locate(45,28);
  gprintf(WHITE,"%c",ans);
  if ((ans == 'Y') || (ans == 'y'))
  {
   on = true;
  }


  cleardevice();
  _setcursortype(_NORMALCURSOR);
  rectangle(0,7,getmaxx(),getmaxy()-15);
  locate(35,0);
  gprintf(LIGHTCYAN," ITERATION ");
  locate(5,1);
  gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
  cin >> n;
  matrix(35,0,"M",g," ITERATION ");
  matrix(35,0,"K",s," ITERATION ");
  mode = 1;
  iter = 0;


  for( i=0; i<n; i++)
  {
   for( j=0; j<n; j++)
   {
    d[i][j] = 0;
   }
   d[i][i] = 1;
   x[i] = (i+1.)/n;
  }
```

```
  icholeski(n, s, q, qi);


  for( i=0; i<n; i++)
  {
   for( j=0; j<n; j++)
   {
    qit[i][j] = qi[j][i];
   }
  }


  improduct(n, qit, g, qitg);
  improduct(n, qi, qitg, a);


aa:
  improduct(n, a, d, ad);


  oev = 0;
  vproduct(n, ad, x, p);
  ev = p[n-1];


  for( i=0; i<n; i++)
  {
   p[i] = p[i]/p[n-1];
  }


  while (fabs(ev-oev) > tol)
  {
   oev = ev;
   iter = iter+1;
   for( i=0; i<n; i++)
   {
    x[i] = p[i];
   }
   if (iter < itmax)
   {
    vproduct(n, ad, x, p);
    ev = p[n-1];
    for( i=0; i<n; i++)
    {
     p[i] = p[i]/p[n-1];
    }
   }
  }
```

```
if (on)
{
 sum = 0;
 for( i=0; i<n; i++)
 {
  sum = sum+p[i]*p[i];
 }
 den = sqrt(sum);
 for( i=0; i<n; i++)
 {
  p[i] = p[i]/den;
 }
}


cleardevice();
_setcursortype(_NORMALCURSOR);
rectangle(0,7,getmaxx(),getmaxy()-15);
locate(35,0);
gprintf(LIGHTCYAN," ITERATION ");
locate(5,2);
gprintf(LIGHTGREEN,"MODE %d ",mode);
locate(5,4);
gprintf(LIGHTCYAN,"Eigenvalue  is: %.4e",ev);
locate(5,6);
gprintf(LIGHTCYAN,"Eigenvector is: %.4e",p[0]);
for( i=1; i<n; i++)
{
 locate(21,6+i);
 gprintf(LIGHTCYAN,"%.4e",p[i]);
}


if (mode < n)
{
 locate(1,27);
 gprintf(YELLOW,"Do you want the next mode? (Y/N) ");
 ans = getch();
 locate(34,27);
 gprintf(WHITE,"%c",ans);
 if ((ans == 'Y') || (ans == 'y'))
 {
  mode = mode+1;
  deflate(n, g, p, d);
  goto aa;
 }
}
locate(1,28);
gprintf(YELLOW,"Another problem? (Y/N) ");
choice = getch();
locate(24,28);
gprintf(WHITE,"%c",choice);
}
```

```
 while ((choice == 'Y') || (choice == 'y'));
}


void icholeski(int n, float w[20][20], float q[20][20], float qi
[20][20])


/*  DECOMP OF MATRIX W SO THAT [QT]*[Q]=[W]
    RETURN [QI](INVERSE OF [QI]) AND [QIT]
    TRANSPOSE OF [QI]                        */


{
 int i,j,k,m;
 int im1,ip1;
 float sum;


 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   q[i][j] = 0;
   qi[i][j] = 0;
  }
 }


 q[0][0] = sqrt(w[0][0]);


 for( j=1; j<n; j++)
 {
  q[0][j] = w[0][j]/q[0][0];
 }


 for( i=1; i<n; i++)
 {
  im1 = i-1;
  sum = 0;
  for( k=0; k<=im1; k++)
  {
   sum = sum+pow(q[k][i],2);
  }
  q[i][i] = sqrt(w[i][i]-sum);
  ip1 = i+1;
  for( j=ip1; j<n; j++)
  {
   sum = 0;
   for( k=0; k<=im1; k++)
   {
    sum = sum+q[k][i]*q[k][j];
```

```
    }
    q[i][j] = 1.*(w[i][j]-sum)/q[i][i];
   }
  }

/* COMPUTE INVERSE OF [Q]     */

  for( i=0; i<n; i++)
  {
   qi[i][i] = 1./q[i][i];
  }

  for( m=1; m<n; m++)
  {
   for( j=m; j<n; j++)
   {
    i = j-m;
    ip1 = i+1;
    sum = 0;
    for( k=ip1; k<=j; k++)
    {
     sum = sum+q[i][k]*qi[k][j];
    }
    qi[i][j] = -sum/q[i][i];
   }
  }


}


void deflate(int n, float g[20][20], float p[20], float d[20]
[20])

/*     USES GRAM-SCHMIDT ORTHONORMALIZATION
       FOR MATRIX DEFLATION                          */

{
 den = 0;
 for( i=0; i<n; i++)
 {
  den = den+p[i]*p[i]*g[i][i];
 }
 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   d[i][j] = d[i][j]-p[i]*p[j]*g[j][j]/den;
  }
 }
}
```

```
void improduct(int n, float u[20][20], float w[20][20], float z
[20][20])
/*  RETURNS X, WHERE X=U*W  */
{
 int i,j,k;
 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   z[i][j] = 0;
   for( k=0; k<n; k++)
   {
     z[i][j] = z[i][j]+u[i][k]*w[k][j];
   }
  }
 }

}

void vproduct(int n, float ad[20][20], float x[20], float p[20])
/*    RETURNS P WHERE P=AD*X
      AD IS AN N*N MATRIX, P & X ARE VECTORS        */
{
 for( i=0; i<n; i++)
 {
  p[i] = 0;
  for( j=0; j<n; j++)
  {
   p[i] = p[i]+ad[i][j]*x[j];
  }
 }
}
```

ภาคผนวก จ.

FILE CHOLJAC.CPP

```
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>


#define false 0
#define true 1


typedef unsigned char logical;


extern int cdecl gprintf(char color, char *fmt, ... );
extern void locate(int x, int y);
extern void matrix(int col, int row, char *ch, float t[20][20],
char *head);
extern void prnt_matrix(int col, int row, char *ch, float x[20]
[20], char *head);
extern void prnt_roots(int col, int row, char *head, char
*detail);


extern int i,j,k,n,itmax;
extern char ans,choice;
extern float g[20][20],s[20][20],a[20][20],qi[20][20],w[20][20];
extern float q[20][20],x[20][20],e[20];
extern float sum,den,tol;


int  ip1,im1;
float qits[20][20];
float u[20][20];


void check_a();
void ccholeski(int n, float w[20][20], float q[20][20], float qi
[20][20]);
void jacobi(int n, float w[20][20], float x[20][20], float e[20],
int itmax,float tol);
void cmproduct(int n, float u[20][20], float w[20][20], float x
[20][20]);
void prnt_eigenvalue();
```

```
void steps(int n, float g[20][20], float s[20][20], float x[20]
[20], float e[20],int itmax, float tol, float a[20][20], float w
[20][20]);


void cmain()
{
Cstart:
 cleardevice();
 _setcursortype(_NORMALCURSOR);
 rectangle(0,7,getmaxx(),getmaxy()-15);
 locate(36,0);
 gprintf(LIGHTCYAN," CHOLJAC ");
 locate(2,1);
 gprintf(WHITE,"This program will determine the eigenvalues and
eigenvectors of a system.");
 locate(2,2);
 gprintf(WHITE,"Input the order, N (20 max), and the [M] and [K]
matrices. Optionally,the");
 locate(2,3);
 gprintf(WHITE,"matrix product [M]*[K], where [M] and [K] are any
two square matrices of");
 locate(2,4);
 gprintf(WHITE,"order N, or the eigenvalues and eigenvectors of
the dynamic matrix, [A] ");
 locate(2,5);
 gprintf(WHITE,"(input as [M]), can be determined.");
 locate(10,7);
 gprintf(LIGHTGREEN,"(1) Matrix product [M]*[K]");
 locate(10,8);
 gprintf(LIGHTGREEN,"(2) Eigenvalues and eigenvectors of [M]");
 locate(10,9);
 gprintf(LIGHTGREEN,"(3) Eigenvalue problem [M]u\"+[K]u=0");
 locate(10,10);
 gprintf(LIGHTGREEN,"(4) Quit");
 locate(10,12);
 gprintf(YELLOW,"Enter selection (1-4): ");
 choice = getch();
 locate(24,12);
 gprintf(LIGHTGREEN,"%c",choice);
 cleardevice();


 itmax = 50;
 tol = 1e-6;


 switch(choice)
 {
  case '1':
       _setcursortype(_NORMALCURSOR);
       rectangle(0,7,getmaxx(),getmaxy()-15);
       locate(28,0);
       gprintf(LIGHTCYAN," MATRIX PRODUCT  [M]*[K] ");
```

```
        locate(5,1);
        gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
        cin >> n;
        matrix(28,0,"M",g," MATRIX PRODUCT  [M]*[K] ");
        matrix(28,0,"K",s," MATRIX PRODUCT  [M]*[K] ");
        for( i=0; i<n; i++)
        {
      for( j=0; j<n; j++)
      {
       u[i][j] = g[i][j];
       w[i][j] = s[i][j];
      }
        }
        cmproduct(n, u, w, x);
        cleardevice();
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(28,0);
        gprintf(LIGHTCYAN," MATRIX PRODUCT  [M]*[K] ");
        locate(5,1);
        gprintf(LIGHTGREEN,"Matrix [A]: ");
        prnt_matrix(28,0,"",x," MATRIX PRODUCT  [M]*[K] ");
        locate(1,28);
        gprintf(YELLOW,"Press Any Key to Continue ");
        getch();


        goto Cstart;
//      break;


    case '2':
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(22,0);
        gprintf(LIGHTCYAN," EIGENVALUES AND EIGENVECTORS OF [M]
");
        locate(5,1);
        gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
        cin >> n;
        matrix(22,0,"A",g," EIGENVALUES AND EIGENVECTORS OF [M]
");
        check_a();
        for( i=0; i<n; i++)
        {
      for( j=0; j<n; j++)
      {
       w[i][j] = g[i][j];
       x[i][j] = 0;
      }
      x[i][i] = 1;
        }
```

```
        jacobi(n , w, x, e, itmax, tol);
        cleardevice();
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(22,0);
        gprintf(LIGHTCYAN," EIGENVALUES AND EIGENVECTORS OF [M]
");
        locate(5,1);
        gprintf(LIGHTGREEN," Eigenvalues of [A] are: ");
        prnt_roots(22,0," EIGENVALUES AND EIGENVECTORS OF [M] ",
              " Eigenvalues of [A] are: ");
        locate(1,28);
        gprintf(YELLOW,"Press Any Key to Continue ");
        getch();
//      printf("\n\n  Eigenvectors of [A] are: \n");
        cleardevice();
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(33,0);
        gprintf(LIGHTCYAN," EIGENVECTORS ");
        locate(5,1);
        gprintf(LIGHTGREEN,"Eigenvectors of [A] are: ");
        prnt_matrix(33,0,"",x," EIGENVECTORS ");
        locate(1,28);
        gprintf(YELLOW,"Press Any Key to Continue ");
        getch();


        goto Cstart;
//      break;


    case '3':
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(24,0);
        gprintf(LIGHTCYAN," EIGENVALUE PROBLEM [M]U\"+[K]U=0 ");
        locate(5,1);
        gprintf(LIGHTGREEN,"Enter Matrix order (Not over 20) : ");
        cin >> n;
        matrix(24,0,"M",g," EIGENVALUE PROBLEM [M]U\"+[K]U=0 ");
        matrix(24,0,"K",s," EIGENVALUE PROBLEM [M]U\"+[K]U=0 ");
        steps(n, g, s, x, e, itmax, tol, a, w);
//        printf("\n\n  Dynamic matrix [A] \n");
        cleardevice();
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(24,0);
        gprintf(LIGHTCYAN," EIGENVALUE PROBLEM [M]U\"+[K]U=0 ");
        locate(5,1);
        gprintf(LIGHTGREEN,"Dynamic Metrix [A] : ");
        prnt_matrix(24,0,"",a," EIGENVALUE PROBLEM [M]U\"+[K]U=0
");
        locate(1,28);
```

```
        gprintf(YELLOW,"Press Any Key to Continue ");
        getch();
        cleardevice();
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(24,0);
        gprintf(LIGHTCYAN," EIGENVALUES AND EIGENVECTORS OF [M]
");
        locate(5,1);
        gprintf(LIGHTGREEN," Eigenvalues are: ");
        prnt_roots(24,0," EIGENVALUES AND EIGENVECTORS OF [M] ",
              " Eigenvalues are: ");
        for( j=0; j<n; j++)
        {
      for( i=0; i<n; i++)
      {
       x[i][j] = x[i][j]/x[n-1][j];
      }
        }
        locate(1,28);
        gprintf(YELLOW,"Do you want orthonormal eigenvectors?
(Y/N) ");
        ans = getche();
        locate(45,28);
        gprintf(WHITE,"%c",ans);
        if ((ans == 'Y') || (ans == 'y'))
        {
      for( j=0; j<n; j++)
      {
       sum = 0;
       for( i=0; i<n; i++)
       {
        sum = sum+x[i][j]*x[i][j];
       }
       den = sqrt(sum);
       for( i=0; i<n; i++)
       {
        x[i][j] = x[i][j]/den;
       }
      }
        }
        cleardevice();
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(33,0);
        gprintf(LIGHTCYAN," EIGENVECTORS ");
        locate(5,1);
        gprintf(LIGHTGREEN," Eigenvectors of [A] are: ");
        prnt_matrix(33,0,"",w," EIGENVECTORS ");
        locate(1,28);
        gprintf(YELLOW,"Press Any Key to Continue ");
        getch();
//        printf("\n  Eigenvectors of [A] are: \n");
//        prnt_matrix(w);
```

```
        cleardevice();
        _setcursortype(_NORMALCURSOR);
        rectangle(0,7,getmaxx(),getmaxy()-15);
        locate(30,0);
        gprintf(LIGHTCYAN," ACTUAL EIGENVECTORS ");
        locate(5,1);
        gprintf(LIGHTGREEN," Actual eigenvectors are: ");
        prnt_matrix(30,0,"",x," ACTUAL EIGENVECTORS ");
        locate(1,28);
        gprintf(YELLOW,"Press Any Key to Continue ");
        getch();


//        printf("\n  Actual eigenvectors are: \n");
//        prnt_matrix(x);


        goto Cstart;
//    break;


    case '4':
//    exit(1);
    break;
   }
}


void check_a()
{
 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   if (g[i][j] != g[j][i])
   {
    printf("\n  [A] must be symmetric ");
    matrix(22,0,"A",g," EIGENVALUES AND EIGENVECTORS OF [M] ");
    check_a();
   }
  }
 }

}


void ccholeski(int n, float w[20][20], float q[20][20], float qi
[20][20])


/*  DECOMP OF MATRIX W SO THAT [QT]*[Q]=[W]
    RETURN [QI](INVERSE OF [QI]) AND [QIT]
    TRANSPOSE OF [QI]                        */
```

```
{
 int i,j,k,m;
 int im1,ip1;
 float sum;

 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   q[i][j] = 0;
   qi[i][j] = 0;
  }
 }

 q[0][0] = sqrt(w[0][0]);

 for( j=1; j<n; j++)
 {
  q[0][j] = w[0][j]/q[0][0];
 }

 for( i=1; i<n; i++)
 {
  im1 = i-1;
  sum = 0;
  for( k=0; k<=im1; k++)
  {
   sum = sum+pow(q[k][i],2);
  }
  q[i][i] = sqrt(w[i][i]-sum);
  ip1 = i+1;
  for( j=ip1; j<n; j++)
  {
   sum = 0;
   for( k=0; k<=im1; k++)
   {
    sum = sum+q[k][i]*q[k][j];
   }
   q[i][j] = 1.*(w[i][j]-sum)/q[i][i];
  }
 }


/* COMPUTE INVERSE OF [Q]     */

 for( i=0; i<n; i++)
 {
  qi[i][i] = 1./q[i][i];
 }

 for( m=1; m<n; m++)
 {
  for( j=m; j<n; j++)
```

```
  {
   i = j-m;
   ip1 = i+1;
   sum = 0;
   for( k=ip1; k<=j; k++)
   {
    sum = sum+q[i][k]*qi[k][j];
   }
   qi[i][j] = -sum/q[i][i];
  }
 }

}



void jacobi(int n, float w[20][20], float x[20][20], float e[20],
int itmax,
          float tol)
/*   SOLVES FOR EIGENVALUES AND RETURNS THEM AS [E]
     RETURNS EIGENVECTORS AS ROWS OF [W]              */
{
 int ip,iter,mc,mr,nm1;
 float amax,drc,f1,f2,fcos,fsin,ftan,g1;


 iter = 1;

/* FIND THE LARGEST(ABS VALUE) OFF-DIAGAONAL TERM IN [A] */

 do
 {
  amax = 0;
  nm1 = n-1;
  for( i=0; i<nm1; i++)
  {
   ip1 = i+1;
   for( j=ip1; j<n; j++)
   {
    if (fabs(w[i][j]) >= amax)
    {
     amax = fabs(w[i][j]);
     mr = i;
     mc = j;
    }
   }
  }

/*    FORMULATE ROTATIONAL MATRIX    */
  if (iter == 1)
  {
   f1 = amax*tol;
  }
  if (amax < f1)
```

```
  {
   goto aa;
  }


  drc = w[mr][mr]-w[mc][mc];
  ftan = (-drc+sqrt(pow(drc,2)+4*pow(amax,2)))/(2*w[mr][mc]);
  fcos = 1./sqrt(1+pow(ftan,2));
  fsin = fcos*ftan;
  for( i=0; i<n; i++)
  {
   g1 = x[i][mr];
   x[i][mr] = fcos*g1+fsin*x[i][mc];
   x[i][mc] = fcos*x[i][mc]-fsin*g1;
  }
  i = 0;

/*   APPLY ROTATIONAL MATRIX TO ZERO LARGEST O-D TERM     */

  while (i != mr)
  {
   f2 = w[i][mr];
   w[i][mr] = fcos*f2+fsin*w[i][mc];
   w[i][mc] = fcos*w[i][mc]-fsin*f2;
   i = i+1;
  }
  i = mr+1;
  while (i != mc)
  {
   f2 = w[mr][i];
   w[mr][i] = fcos*f2+fsin*w[i][mc];
   w[i][mc] = fcos*w[i][mc]-fsin*f2;
   i = i+1;
  }
  i = mc+1;
  while (i <= n)
  {
   g1 = w[mr][i];
   w[mr][i] = fcos*g1+fsin*w[mc][i];
   w[mc][i] = fcos*w[mc][i]-fsin*g1;
   i = i+1;
  }
  f2 = w[mr][mr];
  w[mr][mr] = f2*pow(fcos,2)+w[mr][mc]*2*fcos*fsin+w[mc][mc]*pow
(fsin,2);
  w[mc][mc] = w[mc][mc]*pow(fcos,2)+f2*pow(fsin,2)-w[mr]
[mc]*2*fcos*fsin;
  w[mr][mc] = 0;
  iter = iter+1;
 }
 while (iter < itmax);


aa:
```

```
 for( i=0; i<n; i++)
 {
  e[i] = w[i][i];
 }


}
void cmproduct(int n, float u[20][20], float w[20][20], float x
[20][20])
/*  RETURNS X, WHERE X=U*W  */
{
 int i,j,k;


 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   x[i][j] = 0;
   for( k=0; k<n; k++)
   {
     x[i][j] = x[i][j]+u[i][k]*w[k][j];
   }
  }
 }

}



void steps(int n, float g[20][20], float s[20][20], float x[20]
[20], float e[20],
        int itmax, float tol, float a[20][20], float w[20][20])
{
 float u[20][20];


 ccholeski(n, g, q, qi);


 for( j=0; j<n; j++)
 {
  for( k=0; k<n; k++)
  {
   u[j][k] = qi[k][j];
   w[j][k] = s[j][k];
  }
 }


 cmproduct(n, u, w, x);


 for( j=0; j<n; j++)
```

```
 {
  for(k=0; k<n; k++)
  {
   qits[j][k] = x[j][k];


/*  PRODUCT OF QI TRANSPOSE AND [S]  */

   u[j][k] = x[j][k];
   w[j][k] = qi[j][k];
  }
 }


 cmproduct(n, u, w, x);


 for( j=0; j<n; j++)
 {
  for( k=0; k<n; k++)
  {
   a[j][k] = x[j][k];      /* MATRIX [A] (SEE TEXT)  */
   w[j][k] = x[j][k];
   x[j][k] = 0;
  }
  x[j][j] = 1;
 }


 jacobi(n, w, x, e, itmax, tol);


 for( i=0; i<n; i++)
 {
  for( j=0; j<n; j++)
  {
   u[i][j] = qi[i][j];
   w[i][j] = x[i][j];
  }
 }

 cmproduct(n, u, w, x);

}
```

REFERENCES

1. Burden, R.L., J.D. Faires and A.C. Reynolds (1978)

"Numerical Analysis : Second Edition," Prindle, Weber & Schmidt, Boston, 598 pp.

2. Dowding III, C.H. (1971)

"Response of Building to Ground Vibrations Resulting from Construction Blasting," Ph.D. Dissertation, University of Illinois at Urbana-Champaign, 204 pp.

3. Dowding III, C.H. (1985)

"Blast Vibration Monitoring and Control," Prentice-Hall, Inc., New Jersey, 297 pp.

4. Dowding III, C.H. and P.G. Corser (1981)

"Cracking and Construction Blasting : Importance of Frequency and Free Response," Journal of the Construction Division, ASCE, Vol. 107, No.C01, March 1981, pp. 89 - 106.

5. Hudson, D.E. and G.W. Housner (1957)

"Structural Vibrations Produced by Ground Motion," Transactions, American Society of Civil Engineers, Vol. 122, No. EM 4, Proc. paper No. 2880, pp. 705 - 721.

6. Langefors U., H. Westerberg and B. Kihlstrom (1958)

"Ground Vibrations in Blasting," Water Power, Vol. 10 Part I, September 1958, pp. 335 - 338, 351; Part II, October 1958, pp. 390 - 395; Part III, November 1958, pp. 421 - 424.

7. Medearis, K. (1978)

"Rational Damage Criteria for Low-Rise structures Subjected to Blasting Ground Motions," Pit and Quarry Journal, Vol. 70, No. 5

8. Meirovitch, L. (1986).

"Elements of vibration Analysis ; Second Edition," McGraw-Hill Book Company, 560 pp.

9. Meirovitch, L. (1992)

"Dynamics and Control of Structures," John Wiley & Sons, New York, 425 pp.

10. Newmark, N.M. and W.J. Hall (1969)

"Seismic Design Criteria for Nuclear Reactor Facitlities," Proceedings of the Fourth World Conference on Earthquake Engineering, International Association of Earthquake Engineering, Vol. II, Santiago, Chile, pp. B4, 37 - 50.

11. Rinehart, J.S. (1975)

"Stress Transients in Solid," Hyper-Dynamics, Santa Fe, New Mexico, 230 pp.

12. Selberg, H.L. (1952)

"Transient Compression Wave from Spherical and Cylindrical Cavity, Arkiv för Fysik Journal, Vol. 5, pp. 97 - 108"

13. Timoshenko, S., D.H. Young, W. Weaver, Jr. (1974)

"Vibration Problems in Engineering: Fourth Edition," John Wiley & Sons, New York, 521 pp.

14. Thomson, W.T. (1993)

"Theory of Vibration with Applications : Fourth Edition," Prentice-Hall International, Inc., 546 pp.

15. Veletsos, A.S. and N.M. Newmark (1964)

"Response Spectra of Single Degree-of-Freedom Elastic and Inelastic Systems," Vol. III of Design Procedures for Shock Isolation Systems of Underground Protective Structures, Prepared for the Air Force Weapons Laboratory, Technical Documentary Report RTD TDR-3-3096, Contract No. A.F. 29 (601 - 4565).