ชุดคำสั่งไพทอนสำหรับวิธีไฟไนต์เอลิเมนต์ใน 2 มิติ

Computational framework for the finite element method in 2-D problem with python

พงศ์ลิขิต พินิจจันทร์
สุรชัย เยือกเย็น
เอกชัย เดชะ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมเครื่องกล หลักสูตรวิชาวิศวกรรมเครื่องกล
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2565

Computational framework for the finite element method in 2-D problem with python

Ponglikit Pinijchan
Surachai Yeakyen
Ekachai Decha

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
BACHELOR OF ENGINEERING IN MECHANICAL ENGINEERING
SCHOOL OF ENGINEERING
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2022

ปริญญานิพนธ์ปีการศึกษา 2565

ภาควิชาวิศวกรรมเครื่องกล หลักสูตรวิศวกรรมเครื่องกล

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ชุดคำสั่งไพทอนสำหรับวิธีไฟไนต์เอลิเมนต์ใน 2 มิติ

Computational framework for the finite element method in 2-D problem with python

ผู้จัดทำ

1. นายพงศ์ลิขิต พินิจจันทร์ รหัสประจำตัว 62010583
2. นายสุรชัย เยือกเย็น รหัสประจำตัว 62010982
3. นายเอกชัย เดชะ รหัสประจำตัว 62011077

_____อาจารย์ที่ปรึกษา

(ดร.บำรุง พ่วงเกิด)

ชุดคำสั่งไพทอนสำหรับวิธีไฟไนต์เอลิเมนต์ใน 2 มิติ

นายพงศ์ลิขิต พินิจจันทร์ 62010583
นายสุรชัย เยือกเย็น 62010982
นายเอกชัย เดชะ 62011077
ดร.บำรุง พ่วงเกิด อาจารย์ที่ปรึกษา
ปีการศึกษา 2565

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้นำเสนอการออกแบบและสร้างซอฟแวร์ที่ช่วยในการแก้ปัญหาวิศกรรมเครื่องกลด้วยวิธีไฟไนต์เอลิเมนต์ โดยมีวัตถุประสงค์คือ การศึกษาเกี่ยวกับวิธีการไฟไนต์เอลิเมนต์และเพื่อสร้างซอฟแวร์หรือชุดคำสั่งไพทอนเพื่อช่วยในการแก้ปัญหาทางวิศวกรรมด้วยวิธีไฟไนต์เอ ลิเมนต์ โดยใช้ google Colaboratory ในการสร้างชุดคำสั่งไพทอน ซึ่งเป็นการศึกษาและวิเคราะห์ปัญหากลศาสตร์ของแข็งในสภาพความยืดหยุ่นในสองมิติ โดยที่วัตถุที่ทำการศึกษามีลักษณะเป็นรูปร่างอย่างง่ายคือ รูปสี่เหลี่ยม และรูปสามเหลี่ยม

ขั้นตอนการวิเคราะห์ปัญหาด้วยวิธีไฟไนต์เอลิเมนต์ประกอบไปด้วยขั้นตอน 4 ขั้นตอน เริ่มจากขั้นตอนที่ 1 (Pre-processing) เป็นขั้นตอนการกำหนดปัญหาที่แก้รวมถึง การสร้างตาข่าย(mesh) การกำหนดเงื่อนไขขอบ การกำหนดภาระที่มากระทำ เป็นต้น ขั้นตอนที่ 2 (Solver) เป็นขั้นตอนที่นำค่าต่างได้ไปคำนวณหาผลเฉลย ขั้นตอนที่ 3 (Post-processing) เป็นขั้นตอนที่นำผลเฉลยที่ได้มาแสดงผลในรูปแบบต่างๆ เช่น การแสดงการเสียรูปที่โหนดต่างๆ เป็นต้น สุดท้ายในขั้นตอนที่ 4 (Adaptivity) เป็นขั้นตอนเปรียบเทียบค่าความผิดพลาดกับค่าความผิดพลาดที่รับได้ ซึ่งนำผลเฉลยที่ได้ไปเปรียบเทียบกับค่าจากซอฟต์แวร์อื่นว่าความผิดพลาดเกินกว่าที่รับได้หรือไม่ หากเกินกว่าที่รับได้ ให้นำกลับเข้าสู่กระบวนการวิเคราะห์อีกครั้ง

จากการทดสอบและปรับปรุงแก้ไขชุดคำสั่งไพทอนที่สร้างขึ้น ทำให้ได้ชุดคำสั่งไพทอนที่ประกอบไปด้วยฟังก์ชันต่างๆที่จำเป็นสำหรับวิธีไฟไนต์เอลิเมนต์และมีกำกับวิธีการใช้งานเพื่อให้ง่ายต่อการทำความเข้าใจในชุดคำสั่งที่มากขึ้น โดยเมื่อได้ชุดคำสั่งไพทอนที่เป็นพื้นฐานในการวิเคราะห์ปัญหาทางวิศวกรรมด้วยวิธีไฟไนต์เอลิเมนต์ดังกล่าวมาแล้ว ก็สามารถนำไปต่อยอดและพัฒนาต่อไปได้อีกมาก เช่น การสร้างคำสั่งสำหรับการสร้างตาข่ายสำหรับวัตถุรูปร่างต่างๆ คำสั่งในการใส่แรงทั้งแบบคงที่และไม่คงที่ตามตำแหน่ง รวมไปถึงวิธีการแก้ระบบสมการเชิงเส้นวิธีอื่นนอกเหนือจากที่ใช้ในชุดคำสั่ง

Computational framework for the finite element method in 2-D problem with python

Ponglikit Pinijchan 62010583
Surachai Yeakyen 62010982
Ekachai Decha 62011077
Dr.Bumroong Puangkird Advisor

Academic Year 2565

Abstract

This thesis presents software design and development for mechanical engineering problems with finite element methods. The purpose of this thesis is to study the finite element method and develop software or Python codes for solving mechanical engineering problems with the finite element method using Google Colaboratory for coding. While studying and analyzing the mechanics of solids in 2-D elastic deformation, the problem object that composes of basic shapes such as rectangle and triangle shape.

Process of problems analysis with finite element methods contains a few steps to proceed. First step is called “pre-processing” which is a step that we need to define problems. For example, mesh creating, defining boundary conditions, and defining load. Second step is called “Solver” which is a step in which we use those input parameters we define to calculate and give an approximate solution. Third step is called “post-processing” which is a step in which we convert a solution into another form in order to understand the solution of problems. Fourth step or last step is called “adaptivity” which is a step in which we consider an error between our solution and other software solutions if it’s acceptable or not. If errors that occur are unacceptable, then we restart the process.

After testing and adjusting Python codes that are created, Python codes that contain fundamental functions for finite element method and instructions for using codes are acquired. Then, the codes can be developed in several ways such as making

mesh-creating function for some other shape of the problem, load-applying function for uniform-force and nonuniform-force either, and alternative system of linear equations solver other than the one that is used in current codes.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความอนุเคราะห์จาก อาจารย์ บำรุง พ่วงเกิด อาจารย์ผู้ควบคุมดูแลปริญญานิพนธ์ที่ได้กรุณาให้คำปรึกษาชี้แนะแนวทางรวมทั้งปรับปรุงแก้ไขปัญหาข้อบกพร่องต่างๆในการทำวิจัยด้วยความเอาใจใส่เสมอมาซึ่งคณะผู้จัดทำต้องขอขอบพระคุณเป็นอย่างสูง

ขอขอบคุณภาควิชาวิศวกรรมเครื่องกลสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่เอื้อเฟื้อสถานที่รวมถึงอุปกรณ์ที่จำเป็นทำให้ปริญญานิพนธ์นี้สำเร็จได้อย่างสมบูรณ์ คณะผู้วิจัยมีความซาบซึ้งในความกรุณา และขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาสนี้

ขอขอบคุณเพื่อนๆ ผู้ร่วมทำวิจัยทุกคนที่คอยให้กำลังใจ ให้คำปรึกษา เมื่อมีปัญหาช่วยกันคิดแก้ไขปัญหาร่วมกันทำให้งานวิจัยนี้สำเร็จไปได้ด้วยดี

ขอขอบพระคุณ บิดามารดา และครอบครัว ที่ให้กำลังใจและให้การสนับสนุนการศึกษาเล่าเรียนและทำวิจัยของผู้วิจัยมาโดยตลอด และให้ความช่วยเหลือจนทำให้โครงงานครั้งนี้สำเร็จได้ด้วยดี

พงศ์ลิขิต พินิจจันทร์
สุรชัย เยือกเย็น
เอกชัย เดชะ

สารบัญ

หน้า

สารบัญรูป

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของโครงการ

ในปัจจุบันการแก้ปัญหาทางวิศวกรรมมีวิธีการในการแก้ปัญหาหลากหลายวิธี โดยสามารถแบ่งเป็นประเภทใหญ่ๆได้สองประเภท คือ วิธีแบบดั้งเดิม (Classical Methods) และวิธีเชิงตัวเลข (Numerical Methods) โดยวิธีแบบดั้งเดิมเป็นวิธีแก้ปัญหาเกี่ยวกับสมการแบบ closed-form ด้วยมือ ซึ่งคำตอบที่ได้เป็นสมการผลเฉลยหรือคำตอบแม่นตรง แต่วิธีเชิงตัวเลขนั้นเป็นการแก้ปัญหาโดยใช้คอมพิวเตอร์ช่วยในการประมาณค่าต่างๆโดยไม่ได้แก้สมการปัญหาโดยตรงทำให้ได้คำตอบเป็นค่าประมาณ โดยหนึ่งในวิธีเชิงตัวเลขที่เราสนใจและนำมาใช้ก็คือวิธีไฟไนต์เอลิเมนต์(Finite Element Method)

วิธีไฟไนต์เอลิเมนต์เป็นวิธีการประมาณผลเฉลยโดยอาศัยการแบ่งโดเมนของโครงสร้างของปัญหาที่เราสนใจ เป็นองค์ประกอบย่อยที่มีรูปร่างอย่างง่ายขนาดเล็กโดยที่แต่ละส่วนย่อยนั้นประกอบไปด้วยระบบสมการเชิงอนุพันธ์ย่อยต่างๆ หลังจากแก้ระบบสมการเชิงอนุพันธ์ย่อยทั้งหลายแล้วได้ผลเฉลยออกมาเป็นค่าประมาณโดยกระบวนการคำนวณที่ใช้นั้นไม่ใช้การแก้สมการเชิงอนุพันธ์ย่อยโดยตรงแต่อย่างใด หากแต่เป็นการใช้วิธีการเชิงตัวเลขในการประมาณแทน โดยข้อดีของวิธีไฟไนต์เอลิเมนต์คือสามารถนำมาใช้วิเคราะห์ปัญหาที่มีรูปร่างซับซ้อนได้ สามารถใช้ได้กับปัญหาที่หลากหลาย อาทิ ความเค้น การสั่น การถ่ายเทความร้อน และพลศาสตร์ของไหล

แต่เนื่องจากเป็นวิธีที่ต้องใช้คอมพิวเตอร์ช่วยในการวิเคราะห์และคำนวณนอกจากนี้จำเป็นต้องมีซอฟต์แวร์ที่น่าเชื่อถือ ซึ่งซอฟต์แวร์เหล่านั้นมีค่าใช้จ่ายที่สูง ซึ่งเราจึงมีความคิดสร้างซอฟต์แวร์หรือชุดคำสั่งไพทอนในการช่วยแก้ปัญหาด้วยวิธีไฟไนต์เอลิเมนต์เป็นของตัวเองเพื่อการศึกษาเกี่ยวกับวิธีไฟไนต์เอลิเมนต์ที่มากขึ้น ศึกษาการเขียนโปรแกรม สร้างซอฟต์แวร์ และสามารถตัดค่าจ่ายในที่ต้องเสียในการใช้งานซอฟต์แวร์ลิขสิทธิ์เหล่านั้นได้อีกด้วย

1.2 วัตถุประสงค์ของโครงการ

1. เพื่อศึกษาเกี่ยวกับวิธีการไฟไนต์เอลิเมนต์ซึ่งเป็นกระบวนการวิเคราะห์เชิงตัวเลขอย่างหนึ่ง
2. เพื่อสร้างซอฟต์แวร์หรือชุดคำสั่งไพทอนเพื่อช่วยในการแก้ปัญหาทางวิศวกรรมด้วยวิธีการไฟไนต์เอลิเมนต์
3. สร้างชุดคำสั่งที่สามารถนำไปแก้ไขและพัฒนาต่อไปได้ในงานด้านต่างๆที่เกี่ยวข้องกับวิธีไฟไนต์เอลิเมนต์
4. เพื่อจัดทำเป็นคู่มือในการทำความเข้าใจเกี่ยวกับวิธีการไฟไนต์เอลิเมนต์ด้วยชุดคำสั่งไพทอน

1.3 ขอบเขตของการศึกษา

1. ใช้ไพทอนในการเขียนโปรแกรมที่ช่วยคำนวณไฟไนต์เอลิเมนต์
2. ศึกษาและวิเคราะห์ปัญหากลศาสตร์ของแข็งในสภาพความยืดหยุ่นในสองมิติ
3. สามารถหาตำแหน่งที่เปลี่ยนไปของวัตถุ จากชุดคำสั่งไพทอน
4. วัตถุที่สนใจสามารถมีส่วนที่ถูกยึดอยู่กับที่
5. ลักษณะแรงที่กระทำเป็นแรงกระทำภายนอกที่คงตัว

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. เข้าใจกระบวนการไฟไนต์เอลิเมนต์และสามารถนำมาเปลี่ยนเป็นซอฟต์แวร์หรือชุดคำสั่งไพทอนเพื่อช่วยในการแก้ปัญหาทางวิศวกรรม
2. ได้ซอฟต์แวร์หรือชุดคำสั่งไฟไนต์เอลิเมนต์ที่สามารถช่วยในการแก้ปัญหาทางวิศวกรรม
3. ได้ชุดคำสั่งที่สามารถนำไปพัฒนาต่อได้โดยการกำกับวิธีการการทำงานของชุดคำสั่งเพื่อง่ายต่อการนำไปใช้งานต่อไป

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 กล่าวนำ

บทนี้ได้กล่าวถึงทฤษฎีที่เกี่ยวข้องกับปัญหากลศาสตร์ของแข็งในสองมิติที่ใช้วิธีไฟไนต์เอลิเมนต์วิเคราะห์ โดยเริ่มจากทฤษฎีปัญหากลศาสตร์ของแข็ง ซึ่งเป็นปัญหาที่เกิดจากการที่มีแรงภายนอกมากระทำกับวัตถุที่สามารถเสียรูปได้ เช่น แรงกระทำที่ผิวนอกของชิ้นงาน หรือ ปัญหาที่เกิดจากแรงวัตถุเอง เช่น ปัญหาการโก่งตัวของคานขนาดใหญ่อันเนื่องมาจากผลของน้ำหนักคาน โดยแรงเหล่านี้ที่มากระทำต่อวัตถุสามารถก่อให้เกิดความเค้น และความเครียด ขึ้นภายในวัตถุและถ้าหากมีค่าเกินกว่าค่าวิกฤตค่าหนึ่งก็สามารถก่อให้เกิดความเสียรูปอย่างถาวร หรือ ก่อให้เกิดความเสียหายต่อวัตถุได้ ดังนั้นในการออกแบบชิ้นงานต่างๆ จึงจำเป็นต้องทำการคำนวณหาความเค้น และความเครียดที่เกิดขึ้นภายใต้แรงที่กระทำให้มีความใกล้เคียงกับสถาพการใช้งานจริง ต่อมาจะกล่าวถึงทฤษฎีขั้นตอนวิธีไฟไนต์เอลิเมนต์ ซึ่งปัญหาความยืดหยุ่นใน 2 มิติเป็นปัญหาพื้นฐานที่แสดงถึงศักยภาพของวิธีไฟไนต์ได้เป็นอย่างดีทั้งนี้ก็เพราะว่าผลเฉลยแม่นตรง (exact solution) นั้นหาได้ยาก

2.2 ทฤษฎีสภาพยืดหยุ่น

กล่าวถึงการเกิดความเค้นและความเครียดของวัตถุภายใต้สภาวะการกระทำจากแรงภายนอกโดยวัตถุไม่เกิดการเสียรูปอย่างถาวร และเมื่อมีแรงภายนอกมากระทำกับวัตถุก่อให้เกิดแรงภายในวัตถุ ซึ่งสามารถอธิบายแรงดังกล่าวด้วยรูปแบบของแรงกระทำต่อหนึ่งหน่วยพื้นที่ขนาดเล็กมาก ที่เรียกว่า σ ความเค้น (stress) ดังนี้

$$\sigma = \lim_{\Delta A \to 0} \frac{\Delta F}{\Delta A} \qquad (2.1)$$

และสามารถแบ่งความเค้นออกเป็นองค์ประกอบของความเค้นในแนวตั้งฉากกับพื้นที่เรียกว่า ความเค้นตั้งฉาก σ_N (normal stress) และองค์ประกอบของความเค้นในแนวขนานกับพื้นที่เรียกว่าความเค้นเฉือน τ (shear stress) นั่นคือ

$$\sigma_N = \lim_{\Delta A \to 0} \frac{\Delta F_n}{\Delta A} \qquad (2.2)$$

$$\tau = \lim_{\Delta A \to 0} \frac{\Delta F_t}{\Delta A} \qquad (2.3)$$

เมื่อพิจารณาในสองมิติ พบว่า องค์ประกอบของความเค้นในแนวตั้งฉาก และแนวขนานกับพื้นที่สามาแสดงได้ด้วยรูปของเทนเซอร์ความเค้น (stress tensor) ดังนี้

$$\sigma = \begin{bmatrix} \sigma_{xx} & \tau_{xy} \\ \tau_{yx} & \sigma_{yy} \end{bmatrix} \quad (2.4)$$

และเนื่องจากในสภาวะสมดุลวัตถุต้องไม่มีการหมุน ความเค้นเฉือนที่เกิดขึ้นบนผิวด้านตรงกันข้ามต้องมีค่าเท่ากัน กล่าวคือ

$$\tau_{xy} = \tau_{yx} \quad (2.5)$$

จากสมการพบว่าความเค้นเฉือนดังกล่าวมีค่าเท่ากัน ดังนั้นจึงสรุปได้ว่าในสองมิติเหลือความเค้นที่แตกต่างกัน 3 ตัวเท่านั้น และสามารถเขียนในรูปเวกเตอร์ความเค้นได้ว่า

$$\sigma = [\sigma_{xx} \quad \sigma_{yy} \quad \tau_{xy}]^T \quad (2.6)$$

2.2.1 สมการพื้นฐาน

2.2.1.1 สมการเชิงอนุพันธ์

แผ่นบางที่วางอยู่ในระนาบ x-y ภายใต้แรงกระทำในแนวระนาบนั้นเป็นปัญหาความเค้นระนาบ (plane stress problem) ซึ่งถูกครอบคลุมด้วย 2 สมการเชิงอนุพันธ์ย่อย (partial differential equation) คือ

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} = 0 \quad (2.7)$$

และ

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} = 0 \quad (2.8)$$

โดย σ_x และ σ_y แทนความเค้นฉาก(normal stress) ในทิศแกน x และ y ตามลำดับ ส่วน τ_{xy} แทนความเค้นเฉือน (shearing stress) สมการเชิงอนุพันธ์ย่อยทั้งสองข้างต้นบ่งบอกความเป็นจริงที่ว่า ณ ตำแหน่งใดๆบนแผ่นบางนั้นต้องเกิดความสมดุลของแรงในทิศแกน x และ y ตามลำดับเสมอ นั้นหมายถึงว่า หากสามารถแก้สมการเชิงอนุพันธ์ย่อยทั้งสองนี้ได้ ย่อมได้ผลลัพธ์ซึ่งแสดงถึงปรากฏการณ์ที่เกิดขึ้นสำหรับปัญหานั้นๆ

2.2.1.2 สมการที่เกี่ยวข้อง

ความเค้นฉาก σ_x และ σ_y รวมทั้งความเค้นเฉือน τ_{xy} สามารถเขียนให้อยู่ในรูปแบบของความเครียดย่อย (strain components) ตามกฎของฮุค (Hooke's law) ได้ คือ

$$\{\sigma\}_{(3\times1)}=\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{Bmatrix}=\frac{E}{1-\nu^2}\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \frac{1-\nu}{2}\end{bmatrix}\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \tau_{xy}\end{Bmatrix}=[c]_{(3\times3)}\{\varepsilon\}_{(3\times1)} \qquad (2.9)$$

โดย ε_x และ ε_y แทนความเครียดฉาก (normal strain) ในทิศแกน x และ y ตามลำดับ ส่วน γ_{xy} แทนความเครียดเฉือน (shearing strain) เมตริกซ์ [c] แทนเมทริกซ์ ความยืดหยุ่นของวัสดุ (elasticity matrix) ซึ่งขึ้นอยู่กับค่าโมดูลัสของยัง (Young's modulus) และ ค่าอัตราส่วนปัวส์ซง(Poisson's ratio) ความเครียดย่อยเหล่านี้แปรผันไปกับค่าของการเสียรรูป (displacement) u และ v ในทิศแกน x และ y ตามลำดับ คือ

$$\varepsilon_x=\frac{\partial u}{\partial x} \quad ; \quad \varepsilon_y=\frac{\partial v}{\partial y} \quad ; \quad \gamma_{xy}=\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x} \qquad (2.10)$$

ความสัมพันธ์ระหว่างความเค้นย่อย ความเครียดย่อย และค่าของการเสียรูป ทำให้สมการเชิงอนุพันธ์ของปัญหาความเค้นระนาบในหัวข้อย่อยที่แล้วกลายมาเป็น

$$\frac{\partial}{\partial x}\left[\frac{E}{1-\nu^2}\left(\frac{\partial u}{\partial x}+\nu\frac{\partial v}{\partial y}\right)\right]+\frac{\partial}{\partial y}\left[\frac{E}{2(1-\nu)}\left(\frac{\partial u}{\partial y}+\nu\frac{\partial v}{\partial x}\right)\right]=0 \qquad (2.11)$$

$$\frac{\partial}{\partial x}\left[\frac{E}{2(1-\nu)}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]+\frac{\partial}{\partial y}\left[\frac{E}{1-\nu^2}\left(\nu\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right]=0 \qquad (2.12)$$

นั่นคือปัญหาความเค้นระนาบถูกครอบคลุมด้วยสมการเชิงอนุพันธ์ย่อยจำนวน 2 สมการ เพื่อแก้หาค่าการเสียรูป u และ v ในทิศแกน x และ y ตามลำดับ เมื่อทราบ u และ v แล้วจึงสามารถหาค่าความเครียดย่อย ε_x , ε_y , γ_{xy} แล้วจึงหาค่าความเค้นย่อย σ_x , σ_y, τ_{xy} ตามมาได้

2.3 วิธีการถ่วงน้ำหนักเศษตกค้างแบบกาเลอร์คิน

วิธีการถ่วงน้ำหนักเศษตกค้าง (weighted residual method) เป็นเทคนิคที่รู้จักกันโดยแพร่หลายในการหาผลเฉลยแบบประมาณของสมการณ์เชิงอนุพันธ์ย่อยทั้งแบบเชิงเส้นตรง และ ไม่เชิงเส้นตรง โดยเฉพาะอย่างยิ่งเมื่อถูกนำมาประยุกต์ร่วมกับวิธีของกาเลอร์คิน (galerkin method) เพื่อใช้ในการสร้างสมการไฟไนต์เอลิเมนต์ โดยวิธีการถ่วงน้ำหนักเศษตกค้าง สามารถเลือกฟังก์ชันน้ำหนัก (weight function) ที่หลากหลายจากกลุ่มของฟังก์ชันที่เป็นอิสระต่อกันและนอกจากนี้สามารถเลือกฟังก์ชันถ่วงน้ำหนักที่แตกต่างจากฟังก์ชันการประมาณได้ แต่สำหรับวิธีของกาเลอร์คินให้ทำการเลือกใช้ฟังก์ชันถ่วงน้ำหนักที่เหมือนกับฟังก์ชันการประมาณเสมอ

สมมติว่าใช้วิธีการถ่วงน้ำหนักเศษตกค้าง เพื่อหาผลเฉลยแบบประมาณของปัญหาที่ถูกควบคุมด้วยสมการเชิงอนุพันธ์

$$[A]\{u\} = \{f\} \tag{2.13}$$

ในโดเมน Ω และกำหนดให้ u เป็นผลเฉลยแบบแม่นตรงของสมการ (2.13) ที่สอดคล้องกับเงื่อนไขขอบที่กำหนด เริ่มต้นด้วยการสมมติให้ผลเฉลยแบบประมาณ $\tilde{u}_N$ ของสมการ (2.13) ถูกประมาณคล้ายกันกับวิธีของริทซ์ เช่น

$$\tilde{u}_N(x) = \sum_{j=1}^{N} c_j N_j(x) + N_0(x) \tag{2.14}$$

โดยที่ x หมายถึง กลุ่มของตัวแปรอิสระ ส่วน N_0 และ $N_j(x)$ หมายถึง ฟังก์ชันการประมาณดังนั้นเมื่อแทน $\tilde{u}_N$ ลงในสมการ (2.13) ทำให้ได้ผลเฉลยแบบประมาณที่อาจมีความแตกต่างจากผลเฉลยแบบแม่นตรง ซึ่งผลต่างระหว่างผลเฉลยแบบแม่นตรงและผลเฉลยแบบประมาณนี้เราเรียกว่า ฟังก์ชันเศษตกค้าง (residual function, R)

$$\begin{aligned} R &= A(\tilde{u}_N) - f \\ &= A\left(\sum_{j=1}^{N} c_j N_j + N_0\right) - f \neq 0 \end{aligned} \tag{2.15}$$

และการหาค่าคงที่ c_j สำหรับวิธีการถ่วงน้ำหนักเศษตกค้าง เราสามารถทำได้ด้วยการหาปริพันธ์ฟังก์ชันเศษตกค้างที่ถูกถ่วงด้วยฟังก์ชันน้ำหนัก $w_i(x)$ และกำหนดให้มีค่าเท่ากับศูนย์ดังนี้

$$\int_{\Omega} w_i(x)\, R(x, N_j)\, dV = 0 \quad \text{โดย } i = 1,2,3,\ldots,N \tag{2.16}$$

สุดท้ายเป็นการประยุกต์วิธีของการเลอร์คินเข้ากัยสมการ (2.16) ด้วยการเลือกฟังก์ชันน้ำหนักให้เหมือนกับฟังก์ชันการประมาณ ซึ่งได้สมการถ่วงน้ำหนักเศษตกค้างแบบกาเลอร์คิน คือ

$$\int_{\Omega} N_i(x)\, R(x,N_j) dV = 0 \qquad (2.17)$$

2.4 วิธีไฟไนต์เอลิเมนต์

2.4.1 การสร้างสมการไฟไนต์เอลิเมนต์

วิธีการสร้างสมการไฟไนต์เอลิเมนต์ด้วยวิธีการถ่วงน้ำหนักแบบกาเลอร์คิน โดยเริ่มจากการหาสมการควมคุมของปัญหา ซึ่งความถูกต้องของผลเฉลยแบบประมาณขึ้นกับฟังก์ชันการประมาณที่เลือกใช้ และการหาผลเฉลยของปัญหาทำได้ง่ายขึ้นเนื่องจากสามารถคำนวณแยกตามจำนวนเอลิเมนต์ที่ใช้ หากฟังก์ชันการประมาณถูกกำหนดให้อยู่ในรูปแบบง่าย หรือสามารถสร้างด้วยวิธีการอย่างเป็นระบบ และทำให้ผลเฉลยมีค่าความคลาดเคลื่อนต่ำลงเมื่อโดเมนถูกแบ่งโดยใช้เอลิเมนต์จำนวนมาก ซึ่งการสร้างสมการไฟไนต์เอลิเมนต์ สามารถแบ่งเป็น 6 ขั้นตอนดังนี้

ขั้นตอนที่ 1 แบ่งโดเมนของปัญหาออกเป็นโดเมนย่อยขนาดเล็กที่เรียกว่า เอลิเมนต์

ขั้นตอนที่ 2 เลือกฟังก์ชันการประมาณภายในแต่ละเอลิเมนต์ ซึ่งในที่นี้ใช้การประมาณเชิงเส้นตรง โดยที่ฟังก์ชันการประมาณต้องมีสมบัติดังนี้

$$N_i(x_j) = \begin{cases} 1 & i=j \\ 0 & i\neq j \end{cases} \qquad (2.18)$$

และ

$$\sum_{i=1}^{n} N_i(x) = 1 \qquad (2.19)$$

โดย n หมายถึง จำนวนโหนด

ขั้นตอนที่ 3 หาสมการควบคุมของปัญหา และสร้างฟังก์ชันเศษตกค้างของแต่ละเอลิเมนต์ $R^e(\tilde{u})$

ขั้นตอนที่ 4 ทำการหาปริพันธ์ฟังก์ชันถ่วงน้ำหนักเศษตกค้างดังปรากฏในสมการ (2.17) ก่อให้เกิดระบบสมการพีชคณิตสำหรับแต่ละเอลิเมนต์ในรูปแบบ ดังนี้

$$[k]^e \{u\}^e = \{f\}^e \qquad (2.20)$$

หรือ

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & \cdots & k_{1n} \\ k_{21} & k_{22} & k_{23} & \cdots & k_{2n} \\ k_{31} & k_{32} & k_{33} & \cdots & k_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{n1} & k_{n2} & k_{n3} & \cdots & k_{nn} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{Bmatrix} \qquad (2.21)$$

โดยที่ $[k]^e$ เรียกว่าเมทริกซ์ความแข็งของเอลิเมนต์ (element stiffness matrix)

$\{f\}^e$ หมายถึง เวกเตอร์ของแรงที่กระทำที่โหนดของเอลิเมนต์ (element nodal force vector)

$\{u\}^e$ หมายถึง เวกเตอร์ของตัวแปรไม่ทรายค่าที่โหนดของเอลิเมนต์ (element nodal unknown vector)

n หมายถึง จำนวนโหนดของเอลิเมนต์

ขั้นตอนที่ 5 ทำการรวมสมการพีชคณิตสำหรับแต่ละเอลิเมนต์ (2.20) เข้าด้วยกันทั้งหมด ซึ่งก่อให้เกิดระบบสมการพีชคณิตขนานใหญ่สำหรับปัญหา ดังนี้

$$[K]\{u\}=\{F\} \tag{2.22}$$

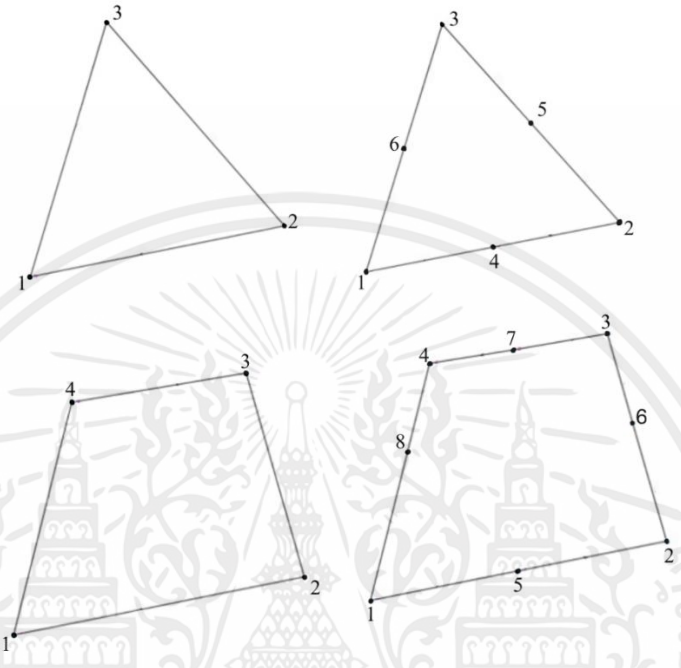
หรือ

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} & \cdots & K_{1n} \\ K_{21} & K_{22} & K_{23} & \cdots & K_{2n} \\ K_{31} & K_{32} & K_{33} & \cdots & K_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{n1} & K_{n2} & K_{n3} & \cdots & K_{nn} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_n \end{Bmatrix} \tag{2.23}$$

แล้วทำการประยุกต์เงื่อนไขขอบของปัญหาเข้ากับสมการ (2.22) หรือ (2.23) ทำให้ได้ระบบสมการพีชคณิตในรูปแบบเมทริกซ์ที่สามารถหาผลเฉลยได้ด้วยวิธีเชิงตัวเลขต่างๆ

2.4.2 ชนิดของเอลิเมนต์

เอลิเมนต์สำหรับปัญหาความเค้นในระนาบสองมิติอาจเป็นเอลิเมนต์สามเหลี่ยม (triangular element) แบบ 3 หรือ 6 โหนด และเอลิเมนต์สี่เหลี่ยมด้านไม่เท่า (quadrilateral element) แบบ 4 หรือ 8 โหนดดังที่แสดงในรูปที่ 2.1

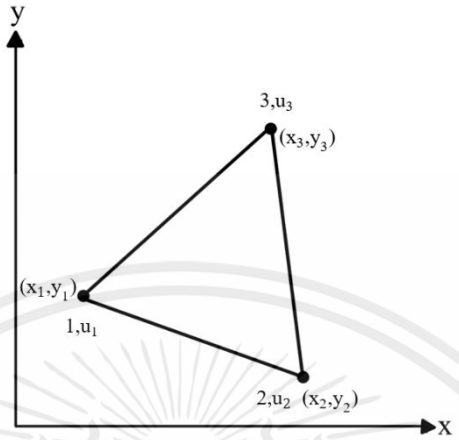


รูปที่ 2.1 เอลิเมนต์สามเหลี่ยมแบบ 3 และ 6 โหนด(บน), เอลิเมนต์สี่เหลี่ยมด้านไม่เท่าแบบ 4 และ 8 โหนด(ล่าง)

เอลิเมนต์ต่างชนิดกันมีฟังก์ชันการประมาณภายใน (interpolation function) ต่างกัน นำมาสู่เมทริกซ์ของความแข็งเกร็ง [K] ที่มีขนาดและค่าที่แตกต่างกันด้วย

2.4.2.1 เอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรง

เอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรงเป็นเอลิเมนต์แบบ 3 โหนด โดยมีการกำหนดลำดับหมายเลขโหนดของเอลิเมนต์เป็นไปในทิศทางแบบทวนเข็มนาฬิกาเสมอดังรูปที่ 2.2



รูปที่ 2.2 เอลิเมนต์สามเหลี่ยมเชิงเส้นตรง

การสร้างฟังก์ชันการประมาณเอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรง เริ่มต้นด้วยการสมมติให้ผลเฉลยมีการเปลี่ยนภายในเอลิเมนต์แบบเชิงระนาบ ซึ่งสามารถเขียนในรูปแบบสมการดังนี้

$$u(x,y)= \alpha_0+\alpha_1 x+\alpha_2 y \tag{2.24}$$

ดังนั้น ผลเฉลยที่โหนดทั้งสาม คือ

$$\begin{aligned} u_{x=x_1,y=y_1} &= u_1=\alpha_0+\alpha_1 x_1+\alpha_2 y_1 \\ u_{x=x_1,y=y_1} &= u_1=\alpha_0+\alpha_1 x_1+\alpha_2 y_1 \\ u_{x=x_1,y=y_1} &= u_1=\alpha_0+\alpha_1 x_1+\alpha_2 y_1 \end{aligned} \tag{2.25}$$

ทำการแก้ระบบสมการเชิงเส้นตรง (2.25) เพื่อหาค่าสัมประสิทธิ์ α_0, α_1 และ α_2 โดยจัดสมการให้อยู่ในรูปแบบเมทริกซ์ แล้วประยุกต์ใช้วิธีการแก้ระบบสมการเชิงเส้นตรงด้วยกฎของเครเมอร์ (Cramer's rule) ได้ดังนี้

$$\begin{aligned} \alpha_0 &= \frac{1}{2A}\left[(x_2y_3-x_3y_2)u_1+(x_3y_1-x_1y_3)u_2+(x_1y_2-x_2y_1)u_3\right] \\ \alpha_1 &= \frac{1}{2A}\left[(y_2-y_3)u_1+(y_3-y_1)u_2+(y_1-y_2)u_3\right] \\ \alpha_2 &= \frac{1}{2A}\left[(x_3-x_2)u_1+(x_1-x_3)u_2+(x_2-x_1)u_3\right] \end{aligned} \tag{2.26}$$

โดยที่

$$A=\frac{1}{2}\left[(x_1y_2-x_2y_1)+(x_3y_1-x_1y_3)+(x_2y_3-x_3y_2)\right] \tag{2.27}$$

เมื่อแทนค่าสมการ (2.26) ลงในสมการ (2.24) ได้การประมาณฟังก์ชัน $u(x,y)$ ของเอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรง ดังนี้

$$u(x,y)=N_1u_1+N_2u_2+N_3u_3 \tag{2.28}$$

และ

$$\begin{aligned} N_1&=\frac{1}{2A}(a_1+b_1x+c_1y)\\ N_2&=\frac{1}{2A}(a_2+b_2x+c_2y)\\ N_3&=\frac{1}{2A}(a_3+b_3x+c_3y) \end{aligned} \tag{2.29}$$

$$\begin{aligned} a_1&=x_2y_3-x_3y_2 \quad & b_1&=y_2-y_3 \quad & c_1&=x_3-x_2\\ a_2&=x_3y_1-x_1y_3 \quad & b_2&=y_3-y_1 \quad & c_2&=x_1-x_3\\ a_3&=x_1y_2-x_2y_1 \quad & b_3&=y_1-y_2 \quad & c_3&=x_2-x_1 \end{aligned} \tag{2.30}$$

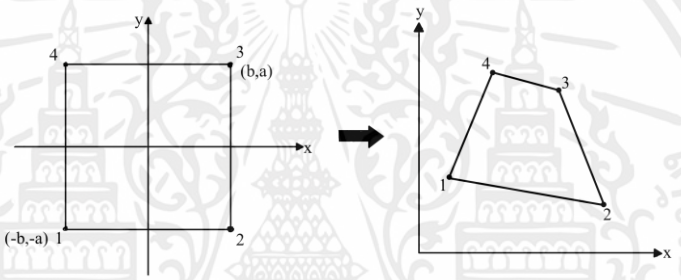
โดยการหาปริพันธ์ของฟังก์ชันการประมาณเอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรงนั้นสามารถทำได้โดยใช้สูตรดังนี้

$$\int_{\Omega^e} N_1^{\alpha}N_2^{\beta}N_3^{\gamma}d\Omega=\frac{\alpha!\beta!\gamma!}{(\alpha+\beta+\gamma+2)!}2A \tag{2.31}$$

2.4.2.2 เอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่

การสร้างฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่ มักนิยมใช้วิธีการส่ง(mapping) บนพิกัดคาร์ทีเซียนจากฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่(bilinear rectangular element) มายังฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่ โดยกำหนดให้เอลิเมนต์รูปสี่เหลี่ยมผืนผ้ามีขนาด $2b \times 2a$ ดังแสดงในรูปที่ 2.3 ซึ่งได้ใช้วิธีการส่งรูปจากรูปทางซ้ายมือมาเป็นรูปทางขวามือ เนื่องจากการสร้างฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่สามารถกระทำได้ง่ายกว่า และเมื่อสมมติให้ผลเฉลยบนเอลิเมต์อยู่ในรูปแบบของระนาบเชิงเส้นตรงดังนี้

$$u(x,y) = \alpha_0 + \alpha_1 x + \alpha_2 y + \alpha_3 xy \tag{2.32}$$



รูปที่ 2.3 การส่งค่าของเอลิเมนต์รูปสี่เหลี่ยมบนพิกัดคาร์ทีเซียน

จากนั้นทำการแทนค่าพิกัดโหนดของเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่ลงในสมการ (2.32) ดังนั้น ผลเฉลยที่โหนดทั้งสี่ มีค่าเท่ากับ

$$\begin{aligned}
u_{x=-b,y=-a} &= u_1 = \alpha_0 - \alpha_1 b - \alpha_2 a + \alpha_3 ab \\
u_{x=b,y=-a} &= u_2 = \alpha_0 + \alpha_1 b - \alpha_2 a - \alpha_3 ab \\
u_{x=b,y=a} &= u_3 = \alpha_0 + \alpha_1 b + \alpha_2 a + \alpha_3 ab \\
u_{x=-b,y=a} &= u_4 = \alpha_0 - \alpha_1 b + \alpha_2 a - \alpha_3 ab
\end{aligned} \tag{2.33}$$

เมื่อแก้สมการ(2.33)เพื่อหาค่าสัมประสิทธิ์ ได้ดังนี้

$$\begin{aligned}
\alpha_0 &= \frac{1}{4}(u_1 + u_2 + u_3 + u_4) \\
\alpha_1 &= -\frac{1}{4b}(u_1 - u_2 - u_3 + u_4) \\
\alpha_2 &= -\frac{1}{4a}(u_1 + u_2 - u_3 - u_4)
\end{aligned} \tag{2.34}$$

$$\alpha_3 = \frac{1}{4ab}(u_1-u_2+u_3-u_4)$$

เมื่อแทนสมการ (2.34) ลงในสมการ (2.32) สามารถแสดงได้ดังนี้

$$\begin{aligned} u(x,y) &= \frac{1}{4}(u_1+u_2+u_3+u_4)-\frac{1}{4b}(u_1-u_2-u_3+u_4)x \\ &\quad -\frac{1}{4a}(u_1+u_2-u_3-u_4)y+\frac{1}{4ab}(u_1-u_2+u_3-u_4)\,xy \\ &= \left(\frac{ab-xa-yb+xy}{4ab}\right)u_1+\left(\frac{ab+xa-yb-xy}{4ab}\right)u_2 \\ &\quad +\left(\frac{1+xa+yb+xy}{4ab}\right)u_3+\left(\frac{1-xa+yb-xy}{4ab}\right)u_4 \\ &= \frac{1}{4ab}[(ab-xa-yb+xy)u_1+(ab+xa-yb-xy)u_2 \\ &\quad +(1+xa+yb+xy)u_3+(1-xa+yb-xy)u_4] \end{aligned} \tag{2.35}$$

และเมื่อจัดอยู่ในรูป

$$u(x,y) = N_1u_1+N_2u_2+N_3u_3+N_4u_4 \tag{2.36}$$

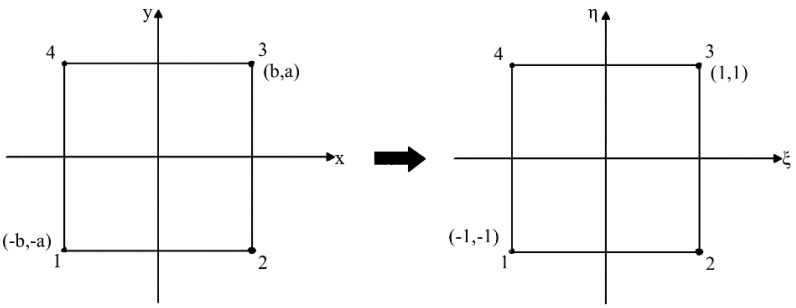
ทำให้ได้ฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่ ดังนี้

$$\begin{aligned} N_1 &= \frac{1}{4ab}(b-x)(a-y)=\frac{1}{4}\left(1-\frac{x}{b}\right)\left(1-\frac{y}{a}\right) \\ N_2 &= \frac{1}{4ab}(b+x)(a-y)=\frac{1}{4}\left(1+\frac{x}{b}\right)\left(1-\frac{y}{a}\right) \\ N_3 &= \frac{1}{4ab}(b+x)(a+y)=\frac{1}{4}\left(1+\frac{x}{b}\right)\left(1+\frac{y}{a}\right) \\ N_4 &= \frac{1}{4ab}(b-x)(a+y)=\frac{1}{4}\left(1-\frac{x}{b}\right)\left(1+\frac{y}{a}\right) \end{aligned} \tag{2.37}$$

จากนั้นทำการแปลงฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่ จากพิกัดคาร์ทีเซียนมายังพิกัดธรรมชาติ (natural coordinate) ดังแสดงในรูปที่ 2.4 ได้ดังนี้

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(1-\eta)\,,\, N_2 = \frac{1}{4}(1+\xi)(1-\eta) \\ N_3 &= \frac{1}{4}(1+\xi)(1+\eta)\,,\, N_4 = \frac{1}{4}(1-\xi)(1+\eta) \end{aligned} \tag{2.38}$$

โดยที่ $-1 \le \xi = \frac{x}{b} \le 1$ และ $-1 \le \eta = \frac{y}{a} \le 1$



รูปที่ 2.4 การแปลงจากพิกัดคาร์ทีเซียนมายังพิกัดธรรชาติ

สุดท้ายเมื่อนำสมการ (2.36) มาประยุกต์ใช้สำหรับเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่ สามารถทำได้โดยการหาความสัมพันธ์ระหว่างตำแหน่งพิกัดโหนดของเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่บนพิกัดคาร์ทีเซียน และตำแหน่งพิกัดโหนดบนพิกัดธรรมชาติ สามารถเขียนในรูปแบบดังที่ปรากฏในสมการ (2.39) และค่าของฟังก์ชันการประมาณตลอดพื้นที่เอลิเมนต์

$$x = \frac{1}{4}[(1-\xi)(1-\eta)x_1+(1+\xi)(1-\eta)x_2+(1+\xi)(1+\eta)x_3+(1-\xi)(1+\eta)x_4]$$

$$y = \frac{1}{4}[(1-\xi)(1-\eta)y_1+(1+\xi)(1-\eta)y_2+(1+\xi)(1+\eta)y_3+(1-\xi)(1+\eta)y_4] \quad (2.39)$$

2.5 การหาปริพันธ์ด้วยวิธีเชิงตัวเลข

การหาปริพันธ์ด้วยวิธีเชิงตัวเลข เป็นอีกหนึ่งเครื่องมือที่มีสำคัญอย่างมากสำหรับวิธีไฟไนต์เอลิเมนต์ โดยเฉพาะอย่างยิ่งเมื่อมีการประยุกต์ใช้เอลิเมนต์อันดับสูง ดังเช่นเอเลเมนต์รูปสามเหลี่ยมกำลังสอง หรือเอลิเมนต์รูปสี่เหลี่ยมผืนผ้าเชิงเส้นคู่ เนื่องจากการหาปริพันธ์เมทริกซ์ของเอลิเมนต์เหล่านี้ในรูปแบบปิด (close form) นั้นเป็นสิ่งที่ยุ่งยากมาก ดังนั้น การหาปริพันธ์ด้วยวิธีเชิงตัวเลขโดยอาศัยสูตรควอเดรเจอร์ของเกาส์-เลอจองด์ (Gauss-Lengendre quadrature หรือ Gauss quadrature) เช่น การหาปริพันธ์ของฟังก์ชันหนึ่งมิติซึ่งเป็นการหาปริพันธ์ที่ง่ายที่สุด โดยอาศัยแนวคิดหลักของควอเดรเจอร์ของเกาส์-เลอจองด์ ก็คือการแทนที่การหาปริพันธ์ด้วยวิธีแม่นตรง โดยอาศัยผลรวมที่เกิดจากการถ่วงน้ำหนักของค่าฟังก์ชัน $f(\xi)$ ณ ตำแหน่งต่างๆในช่วง $-1 \leq \xi \leq 1$ ดังนี้

$$I = \int_{-1}^{1} f(\xi)\, d\xi = \sum_{i=1}^{N} W_I f(\xi_I) \quad (2.40)$$

N	Points, ξ_I	Weights, W_I
1	0.0000000000	2.0000000000
2	± 0.5773502692	1.0000000000
3	0.0000000000	0.8888888889
	± 0.7745966692	0.5555555555
4	± 0.3399810435	0.6521451548
	± 0.8611363116	0.3478548451
5	0.0000000000	0.5688888889
	± 0.5384693101	0.4786286705
	± 0.9061798459	0.2369268850
6	± 0.2386191861	0.4679139346
	± 0.6612093865	0.3607615730
	± 0.9324695142	0.1713244924

รูปที่ 2.5 สูตรควอเดรเจอร์ของเกาส์ในหนึ่งมิติ

เมื่อ N คือ จำนวนตำแหน่งจุดเกาส์ (Gauss point) และ W_I หมายถึง ค่าถ่วงน้ำหนักที่ตำแหน่งจุดเกาส์ ดังแสดงในรูปที่ 2.5 โดยสูตรควอเดรเจอร์ของเกาส์-เลอจองด์ให้ผลเฉลยแบบแม่นตรง เมื่อกำลังของฟังก์ชัน $f(\xi)$ มีค่าน้อยกว่าหรือเท่ากับ $2n-1$ เช่น เมื่อ N = 2 สามารถให้ผลเฉลยแบบแม่นตรงเมื่อใช้กับฟังก์ชันกำลังสามหรือน้อยกว่า เป็นต้น

ส่วนการหาปริพันธ์ของฟังก์ชันสองมิตินั้น สามารถทำได้โดยอาศัยแนวคิดหลักของควอเดรเจอร์ของเกาส์-เลอจองด์ กล่าวคือ การแทนที่การหาปริพันธ์ด้วยวิธีแม่นตรงด้วยผลรวมของการถ่วงน้ำหนักของค่าฟังก์ชัน $f(\xi,\eta)$ ณ ตำแหน่งจุดเกาส์ต่างๆในช่วง $-1 \leq \xi \leq 1$ และ $-1 \leq \eta \leq 1$ ดังนี้

$$I = \int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)\, d\xi d\eta$$
$$= \sum_{j=1}^{N}\sum_{i=1}^{N} W_i W_j f(\xi_i,\eta_j) \qquad (2.41)$$

สมการไฟไนต์เอลิเมนต์ของปัญหาโดยทั่วไป สามารถแสดงไฟไนต์เอลิเมนต์เมทริกซ์ในรูปแบบ เช่น

$$K^e = \int_{\Omega^e} B^T B \, d\Omega \qquad (2.42)$$

โดย B หมายถึงเมทริกซ์แกรเดียนในแนวแกน x และ y ตามลำดับ ดังนี้

$$B = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} \end{bmatrix} \qquad (2.43)$$

การหาปริพันธ์ของเมทริกซ์เกรเดียนต์ดังกล่าวสำหรับเอลิเมนต์บางประเภท เช่น เอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่ นิยมใช้วิธีการหาปริพันธ์ด้วยวิธีเชิงตัวเลข ตัวอย่างเช่น เกรเดียนต์ของ u สำหรับเอลิเมนต์ใดๆที่ประกอบด้วย N โหนด สามารถแสดงในรูปเมทริกซ์ได้ ดังนี้

$$\frac{\partial u}{\partial x} = \sum_{i=1}^{N} \frac{\partial N_i}{\partial x} u_i \ , \frac{\partial u}{\partial y} = \sum_{i=1}^{N} \frac{\partial N_i}{\partial y} u_i \qquad (2.44)$$

สำหรับพจน์ $\frac{\partial N_i}{\partial x}$ และ $\frac{\partial N_i}{\partial y}$ นั้น เราสามารถประยุกต์กฎลูกโซ่เข้ากับพจน์ดังกล่าวเพื่อให้แปลงอยู่ในระบบพิกัดธรรมชาติ (ξ,η) ดังนี้

$$\frac{\partial N_i}{\partial \xi} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \xi}$$

$$\frac{\partial N_i}{\partial \eta} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \eta} \qquad (2.45)$$

หรือสามารถเขียนรูปแบบเมตริกได้ว่า

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix} = [J] \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix} \qquad (2.46)$$

โดย $[J]$ เรียกว่า เมทริกซ์จาโคเบียน (Jacobian matrix) และเนื่องจากตำแหน่งพิกัด (x,y) บนเอลิเมนต์ใดๆสามารถเขียนในรูปแบบของฟังก์ชันการประมาณเอลิเมนต์ในระบบพิกัดธรรมชาติ (ξ,η) ซึ่งเป็นที่มาของเทคนิคการเปลี่ยนแปลงพิกัดระหว่างระบบแกน 2 ระบบ ดังเช่น

$$x = \sum_{i=1}^{N} N_i(\xi,\eta) x_i$$

$$y = \sum_{i=1}^{N} N_i(\xi,\eta) y_i \qquad (2.47)$$

ดังนั้น เมทริกซ์จาโคเบียนในระบบพิกัดธรรมชาติ (ξ,η) มีค่าเท่ากับ

$$[J] = \begin{bmatrix} \sum_{i=1}^{N} \frac{\partial N_i(\xi,\eta)}{\partial \xi} x_i & \sum_{i=1}^{N} \frac{\partial N_i(\xi,\eta)}{\partial \xi} y_i \\ \sum_{i=1}^{N} \frac{\partial N_i(\xi,\eta)}{\partial \eta} x_i & \sum_{i=1}^{N} \frac{\partial N_i(\xi,\eta)}{\partial \eta} y_i \end{bmatrix} \qquad (2.48)$$

และโดยการหาเมทริกซ์จาโคเบียนผกผัน ทำให้สามารถหาเมทริกซ์เกรเดียนต์ นั่นคือ

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} \text{ โดย } i = 1,2 \ldots, N \qquad (2.49)$$

เมื่อแทนสมการ (2.49) ลงในสมการ (2.44) ดังนี้

$$\begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{Bmatrix} = [J]^{-1} \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \cdots & \frac{\partial N_N}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \cdots & \frac{\partial N_N}{\partial \eta} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{Bmatrix} \quad (2.50)$$

และการแปลงพื้นที่ (area transformation) ระหว่างระบบพิกัดแกนทั้งสอง สามารถทำได้โดยใช้สูตรดังนี้

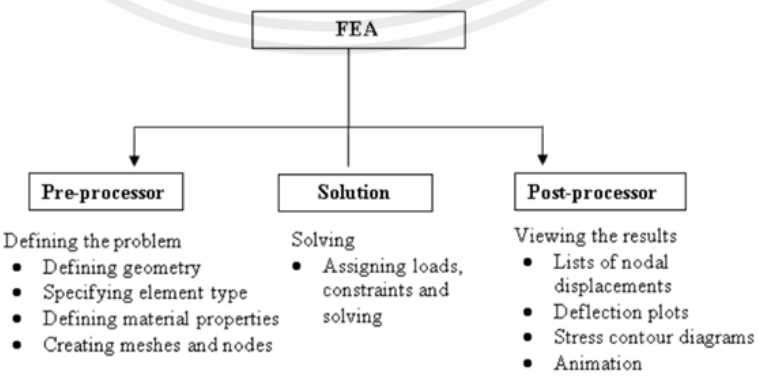
$$dxdy = |J| \partial\xi \partial\eta \quad (2.51)$$

บทที่ 3
การออกแบบโปรแกรมคอมพิวเตอร์ด้วยวิธีไฟไนต์เอลิเมนต์

3.1 กล่าวนำ

วิธีไฟไนต์เอลิเมนต์เป็นวิธีเชิงตัวเลขที่ช่วยในการแก้ปัญหาทางวิศวกรรมด้วยการประมาณผลเฉลยของปัญหาที่กำลังวิเคราะห์โดยอาศัยหลักของการแบ่งโดเมนออกเป็นโดเมนย่อยที่ไม่ทับซ้อนกัน เรียกว่าเอลิเมนต์ (element) จากนั้นจึงทำการสร้างสมการไฟไนต์เอลิเมนต์สำหรับแต่ละเอลิเมนต์ ซึ่งในการวิเคราะห์ได้กำหนดให้ผลเฉลยของแต่ละเอลิเมนต์ถูกจัดเก็บไว้ที่โหนด (node) เมื่อนำเอลิเมนต์ทั้งหมดมารวมกันเป็นโดเมน หรือที่เรียกว่า ตาข่าย (mesh) ทำให้ได้ระบบสมการพีชคณิตที่สามารถหาผลเฉลยได้โดยใช้วิธีเชิงตัวเลขที่เหมาะสม แต่เนื่องจากผลเฉลยที่ได้เป็นผลเฉลยแบบประมาณของสมการเชิงอนุพันธ์ ดังนั้น ความแม่นของผลเฉลยจึงสามารถขึ้นได้กับหลายปัจจัย โดยปัจจัยหลักที่มีผลกระทบสูงก็คือขนาดของเอลิเมนต์ และฟังก์ชันการประมาณเอลิเมนต์ ซึ่งแสดงให้เห็นว่าการกำหนดขนาดของเอลิเมนต์ที่เหมาะสมกับแต่ละปัญหา ผลต่อความแม่นยำและเวลาที่ใช้ในการคำนวณ สามารถสรุปการวิเคราะห์ปัญหาด้วยวิธีไฟไนต์เอลิเมนต์ประกอบด้วย 3 ขั้นตอนหลัก ได้แก่ ขั้นตอนแรกเป็นการแบ่งโดเมนที่ต้องการวิเคราะห์ออกเป็นเอลิเมนต์ย่อยและไม่ทับซ้อนกัน ขั้นตอนที่สอง สร้างสมการไฟไนต์เอลิเมนต์ซึ่งเป็นระบบสมการพีชคณิตสำหรับแต่ละเอลิเมนต์ และขั้นตอนที่สาม เป็นการแก้ระบบสมการพีชคณิตด้วยวิธีเชิงตัวเลขที่เหมาะสมเพื่อหาผลเฉลยแบบประมาณ ซึ่งขั้นตอนทั้งสามได้ถูกนำไปใช้ในการประดิษฐ์เป็นโปรแกรมคอมพิวเตอร์ต่อไป

โดยภาพรวมของการประยุกต์ใช้วิธีไฟไนต์เอลิเมนต์ร่วมกับตัวชี้วัดค่าความผิดพลาด เพื่อให้ได้ผลเฉลยที่มีความแม่นยำสูงในการแก้ปัญหาทางวิศวกรรมนั้น ประกอบด้วย 4 ขั้นตอนหลักแสดงดังรูปที่ 3.1



รูปที่ 3.1 ขั้นตอนการวิเคราะห์ปัญหาด้วยวิธีไฟไนต์เอลิเมนต์

ขั้นตอนที่ 1 (Pre-processing) เป็นขั้นตอนการเตรียมแบบจำลองไฟไนต์เอลิเมนต์จากปัญหาจริง โดยเริ่มจากการเตรียมแบบจำลอง การสร้างตาข่าย (mesh) การกำหนดเงื่อนไขขอบ และการกำหนดภาระที่มากระทำกับแบบจำลอง

ขั้นตอนที่ 2 (Solver) เมื่อแบบจำลองไฟไนต์เอลิเมนต์เสร็จสมบูรณ์แล้ว ข้อมูลทั้งหมดถูกส่งต่อไปที่โปรแกรมคำนวณ ซึ่งเริ่มจากการสร้างสมการไฟไนต์เอลิเมนต์ให้กับแต่ละเอลิเมนต์ กำหนดเงื่อนไขที่ขอบ และระบุภาระที่มากระทำกับเอลิเมนต์ จากนั้นทำการประกอบเอลิเมนต์ทั้งหมดเข้าด้วยกัน ทำให้ได้ระบบสมการพีชคณิตของปัญหาที่สามารถหาผลเฉลยได้ด้วยวิธีเชิงตัวเลขที่กำหนด

ขั้นตอนที่ 3 (Post-processing) เมื่อโปรแกรมได้ผลเฉลย ลำดับถัดไปเป็นการคำนวณหาค่าฟลักซ์ที่โหนด เพื่อการแสดงผลปริมาณทางฟิสิกส์ต่างๆ

ขั้นตอนที่ 4 (Adaptivity) ในกรณีที่ต้องการผลเฉลยที่มีความแม่นสูงขึ้น ให้ทำการเปรียบเทียบค่าตัวชี้วัดค่าความผิดพลาดของโดเมนกับค่าความผิดพลาดที่ยอมรับได้ โดยถ้าหากมีค่าตัวชี้วัดค่าวามผิดพลาดของโดเมนมากกว่าค่าความผิดพลาดที่ยอมรับได้ ให้นำกลับเข้าสู่กระบวนการวิเคราะห์ใหม่อีกครั้ง แต่ในทางตรงข้ามให้สิ้นสุดการทำงานของโปรแกรม

โดยจากขั้นตอนทั้งสี่ขั้นตอนที่กล่าวไว้ข้างต้น เราได้นำไปใช้ในการเขียนชุดคำสั่งไพทอนด้วย google Colaboratory โดยแต่ละชุดคำสั่งจะอธิบายในหัวข้อถัดไป

3.2 ฟังก์ชัน (Function)

3.2.1 ฟังก์ชันผลเฉลย (Solution Function)

ฟังก์ชัน solution() เป็นฟังก์ชันที่ใช้สำหรับหาการเคลื่อนที่ของโครงสร้างหรือองค์ประกอบต่าง ๆ ที่เกิดจากแรงที่กระทำต่อโครงสร้าง โดยฟังก์ชันรับค่าอาร์กิวเมนต์ที่ต้องการดังนี้:

- nDof: จำนวนของ degrees of freedom ทั้งหมดของโครงสร้าง
- fixDof: ลำดับของ degrees of freedom ที่ต้องการถูกกำหนดให้หยุดเคลื่อนที่หรือไม่สามารถเคลื่อนที่ได้ เช่น โครงสร้างที่ถูกติดตั้งบนพื้นดินจะมี degrees of freedom ที่เป็นค่าคงที่เพราะไม่สามารถเคลื่อนที่ได้
- K: เมทริกซ์ความต้านทานของโครงสร้าง ซึ่งแสดงถึงความต้านทานของโครงสร้างที่ต่อกับแรงที่กระทำต่อโครงสร้าง
- force: เวกเตอร์แรงที่กระทำต่อโครงสร้าง

หลังจากได้รับค่าอินพุตแล้ว ฟังก์ชันจะนำข้อมูลที่ได้รับมาใช้ในการหา active degrees of freedom ซึ่งเป็น degrees of freedom ที่ยังไม่ได้ถูกกำหนดเป็นค่าคงที่ หรือไม่ได้รับผลกระทบจากการแรงที่กระทำต่อโครงสร้าง จากนั้นฟังก์ชันจะใช้เมทริกซ์ความต้านทานของโครงสร้าง (K) และเวกเตอร์แรง (force) เพื่อหาผลการเคลื่อนที่ของ active degrees of freedom ด้วยการใช้ np.linalg.solve() หรือคำสั่ง scipy.sparse.linalg.cg() ในการแก้ระบบสมการเชิงเส้น โดยเป็นการใช้วิธี Cholesky Decomposition และ Conjugate gradient ตามลำดับ จากนั้น ฟังก์ชันจะสร้างเวกเตอร์ displacement (disp) ขนาด nDof x 1 ด้วย np.zeros และกำหนดค่า displacement ของ degree of freedom ที่ active ด้วย U ที่ได้จากการแก้ระบบก่อนหน้านี้ และทำการคืนค่า displacement เป็นตัวแปร disp

```
1    import numpy as np
2    def solution(nDof, fixDof, K, force):
3         activeDof = np.setdiff1d(np.arange(nDof),
          fixDof)
4         U = np.linalg.solve(K[np.ix_(activeDof,
          activeDof)],  force[activeDof])
5         disp = np.zeros((nDof, 1))
6         disp[activeDof] = U
7         return disp
```

```
1    import scipy.sparse.linalg
2    def solution(nDof,fixDof,K,force):
3         activeDof = np.setdiff1d(np.arange(nDof),
          fixDof)
4         U, info =
          scipy.sparse.linalg.cg(K[np.ix_(activeDof,activ
          eDof)], force[activeDof])
5         disp = np.zeros(nDof)
6         disp[activeDof] = U
7         return disp
```

คำสั่งที่ 3.1 ฟังก์ชันผลเฉลย

3.2.2 ฟังก์ชันใช้ประมาณค่า (Shape Function)

ฟังก์ชัน shape function เป็นฟังก์ชันใช้ประมาณค่าเอลิเมนต์ตามชนิดของเอลิเมนต์โดย ฟังก์ชัน shapeFunc(xi, eta) ใช้สำหรับคำนวณฟังก์ชันใช้ประมาณค่าเอลิเมนต์และค่าอนุพันธ์ของฟังก์ชันใช้ประมาณเอลิเมนต์ของเอลิเมนต์รูปร่างต่างๆ ซึ่งเป็นฟังก์ชันของตัวแปร xi และ eta ซึ่งเป็นตัวแปรพารามิเตอร์ที่ใช้ในการแทนตำแหน่งของจุดภายใน element

พารามิเตอร์ที่ใช้ในคำสั่ง:

xi: ตัวแปรพารามิเตอร์ที่ใช้ในการแทนตำแหน่งของจุดภายในเอลิเมนต์ในแนวแกน x โดย xi จะมีค่าอยู่ระหว่าง -1 ถึง 1

eta: ตัวแปรพารามิเตอร์ที่ใช้ในการแทนตำแหน่งของจุดภายในเอลิเมนต์ในแนวแกน y โดย eta จะมีค่าอยู่ระหว่าง -1 ถึง 1

ผลลัพธ์:

N: ค่าของฟังก์ชันใช้ประมาณค่าคำนวณได้จากฟังก์ชันและเป็นเวกเตอร์แบบ column matrix ขนาด 4x1 โดยแต่ละแถวจะแทนค่าของฟังก์ชันใช้ประมาณค่าสำหรับแต่ละจุดภายในเอลิเมนต์

dN: ค่าของอนุพันธ์ของฟังก์ชันใช้ประมาณค่าที่คำนวณได้จากฟังก์ชันและเป็น matrix ขนาด 4x2 โดยแต่ละ แถวจะแทนค่าของอนุพันธ์ของฟังก์ชันใช้ประมาณค่าที่แทนด้วย index row สำหรับแต่ละจุดภายในเอลิเมนต์โดย index column จะแทนอนุพันธ์ของตัวแปร xi และ eta ตามลำดับ

ฟังก์ชันการประมาณค่าของเอลิเมนต์แต่ละรูปร่างมีดังนี้

1.ฟังก์ชันการประมาณเอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรง

```
1    import numpy as np
2    def shapeFuncT3(xi, eta):
3    # สร้างเวกเตอร์ขนาด 3x1 ที่มีค่าเริ่มต้นเป็นศูนย์
4    N = np.zeros((3,1))
5
6    # คำนวณค่า shape function ของสามเหลี่ยมเชิงเส้นตรงจากสมการที่กำหนด
7    N[:,0] = np.array([(1-xi-eta), xi, eta])
8
9    # สร้างเมทริกซ์ขนาด 3x2 ที่มีค่าเริ่มต้นเป็นศูนย์
10    dN = np.zeros((3,2))
11
12    # คำนวณค่า gradient vector ของสามเหลี่ยมเชิงเส้นตรงจากสมการที่กำหนด
13    dN[0,:] = np.array([-1, -1])
14   dN[1,:] = np.array([1, 0])
15   dN[2,:] = np.array([0, 1])
16
17    # ส่งค่ากลับเป็น shape function และ gradient vector ของสามเหลี่ยมเชิง
เส้นตรง
18    return N, dN
```

คำสั่งที่ 3.2 ฟังก์ชันการประมาณเอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรง

2.ฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่

```
1    import numpy as np
2     def shapeFuncQ4(xi, eta):
3    # คำนวณฟังก์ชันประมาณค่า (shape function)
4    N = np.array([
5        (1 - xi) * (1 - eta),
6        (1 + xi) * (1 - eta),
7        (1 + xi) * (1 + eta),
8        (1 - xi) * (1 + eta)
9    ]) / 4
10    # คำนวณ gradient ของฟังก์ชันประมาณค่า (shape function)
11    dN = np.array([
12        [-1/4 * (1 - eta), -1/4 * (1 - xi)],
13        [1/4 * (1 - eta), -1/4 * (1 + xi)],
14        [1/4 * (1 + eta), 1/4 * (1 + xi)],
15        [-1/4 * (1 + eta), 1/4 * (1 - xi)]])
16
17    return N, dN
```

คำสั่งที่ 3.3 ฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่

3.ฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมกำลังสอง

```
import numpy as np
 def shapeFuncQ8(xi,eta):
# คำนวณค่าของฟังก์ชันรูปร่าง N
N = np.zeros((8,1))
N[:,0] = 1/4*np.array([(1-xi)*(1-eta)*(-xi-eta-1),
                       (1+xi)*(1-eta)*(xi-eta-1),
                       (1+xi)*(1+eta)*(xi+eta-1),
                       (1-xi)*(1+eta)*(-xi+eta-1),
                      2*(1-xi*xi)*(1-eta),
                       2*(1+xi)*(1-eta*eta),
                      2*(1-xi*xi)*(1+eta),
                      2*(1-xi)*(1-eta*eta)])
# คำนวณค่าของเมทริกซ์เกรเดียนของ N โดยใช้ฟังก์ชันดัชนีอนุพันธ์
dN = np.zeros((8,2))
dN[0,:] = 1/4*np.array([-(1-eta)*(-xi-eta-1)-(1-
 xi)*(1-eta),-(1-xi)*(-xi-eta-1)-(1-xi)*(1-eta)])
dN[1,:] = 1/4*np.array([(1-eta)*(xi-eta-
1)+(1+xi)*(1-eta),-(1+xi)*(xi-eta-1)-(1+xi)*(1-eta)])
dN[2,:] = 1/4*np.array([(1+eta)*(xi+eta-
1)+(1+xi)*(1+eta),(1+xi)*(xi+eta-1)+(1+xi)*(1+eta)])
dN[3,:] = 1/4*np.array([-(1+eta)*(-xi+eta-1)-(1-
 xi)*(1+eta),(1-xi)*(-xi+eta-1)+(1-xi)*(1+eta)])
dN[4,:] = 1/2*np.array([(-2*xi)*(1-eta),-(1-xi*xi)])
dN[5,:] = 1/2*np.array([(1-eta*eta),(1+xi)
 *(-2*eta)])
dN[6,:] = 1/2*np.array([(-2*xi)*(1+eta),(1-xi*xi)])
dN[7,:] = 1/2*np.array([-(1-eta*eta),
 (1-xi)*(-2*eta)])
 return N, dN
```

คำสั่งที่ 3.4 ฟังก์ชันการประมาณเอลิเมนต์รูปสี่เหลี่ยมกำลังสอง

3.2.3 ฟังก์ชันจาโคเบียน (Jacobian Function)

ฟังก์ชัน Jacobian(nodeCoord, nderiv) ใช้สำหรับคำนวณ Jacobian matrix และค่าอนุพันธ์ของตำแหน่ง (derivative) ของจุดภายในเอลิเมนต์ โดยใช้ค่าตำแหน่งของโหนดและค่าอนุพันธ์ของฟังก์ชันการประมาณค่า ซึ่งเป็นฟังก์ชันของตำแหน่ง xi และ eta

อธิบายพารามิเตอร์ที่ใช้ในคำสั่ง:

nodeCoord: ตัวแปรที่นำเข้าที่เป็นเมทริกซ์ขนาด 2x2 ซึ่งแต่ละแถวจะเป็นตำแหน่งของโหนดในระบบพิกัด global coordinate system

nderiv: ตัวแปรที่นำเข้าที่เป็นเมทริกซ์ขนาด 4x2 ซึ่งแต่ละแถวจะเป็นค่าอนุพันธ์ของฟังก์ชันการประมาณค่าต่อ xi และ eta ของจุดภายในเอลิเมนต์ตามลำดับ

ผลลัพธ์:

J: ค่าของจาโคเบียนเมทริกซ์ที่คำนวณได้จากฟังก์ชันและเป็นเมทริกซ์ขนาด 2x2 โดยแต่ละสมาชิกของเมทริกซ์จะเป็นค่าของอนุพันธ์ของตำแหน่งของจุดภายในเอลิเมนต์ต่อ xi และ eta โดยตามลำดับ

xyDeriv: ค่าอนุพันธ์ของตำแหน่ง (derivative) ของจุดภายในเอลิเมนต์ที่คำนวณได้จากฟังก์ชันและเป็นเมทริกซ์ขนาด 2x2 โดยแต่ละสมาชิกของเมทริกซ์จะแทนค่าอนุพันธ์ของตำแหน่ง x และ y ของจุดภายในเอลิเมนต์ตามลำดับ

```
1  import numpy as np
2   def Jacobian(nodeCoord,nderiv):
3    J = np.matmul(np.transpose(nodeCoord),nderiv)
4
5    bT = np.transpose(nderiv)
6    aT = J.T
7    xT = np.linalg.solve(aT,bT)
8    xyDeriv = np.transpose(xT)
9    return J, xyDeriv
```

คำสั่งที่ 3.5 ฟังก์ชันจาโคเบียน

3.1.4 เกาส์ควอดราเจออร์ (Gauss Quadrature)

ฟังก์ชัน gaussQuadrature ใช้สำหรับการคำนวณน้ำหนักและตำแหน่งของจุดเกาส์สำหรับการประมาณค่าอินทิกรัลด้วยวิธีเกาส์เซียนควอดราเจอร์ (Gaussian Quadrature) โดยมีการรับค่าตัวแปร option ซึ่งเป็นตัวกำหนดว่าจะใช้การประมาณค่าด้วยจุดเกาส์ในรูปแบบใด ๆ

- เมื่อ option = 1 จะทำการคำนวณด้วยการใช้จุดเกาส์แบบ full integration โดยจะใช้จุดเกาส์ 4 จุดในพื้นที่สี่เหลี่ยม

- เมื่อ option ไม่ใช่ 1 จะทำการคำนวณด้วยการใช้จุดเกาส์แบบ linear integration โดยจะใช้จุดเกาส์ 4 จุดในพื้นที่สี่เหลี่ยม

- โดยฟังก์ชัน gaussQuadratureQ8 และ gaussQuadratureT3 จะเป็นการคำนวณจุดเกาส์ และน้ำหนักของจุดด้วยวิธีเดียวกันแต่จะแตกต่างกันตามสูตรคำนวณของแต่ละพื้นที่

- ใน gaussQuadratureQ8 จะใช้ option เป็นตัวกำหนดว่าจะใช้การประมาณค่าด้วยจุดเกาส์แบบ full 2nd order integration ที่มีจุดเกาส์ 9 จุด

- หาก option เป็นค่าอื่น ๆ จะใช้ option ในการคำนวณจุดเกาส์แบบ linear integration โดยจะใช้จุดเกาส์ 4 จุด

ใน gaussQuadratureT3 จะเป็นการคำนวณจุดเกาส์สำหรับสามเหลี่ยม T3 โดยใช้ option แบ่งเป็น 4 แบบ คือ

-option 1 ใช้จุดเกาส์ 3 จุด

-option 2 ใช้จุดเกาส์ 3 จุดที่แตกต่างจาก option 1

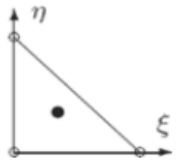
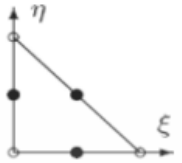
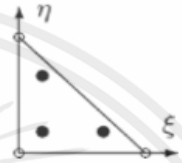
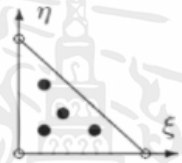
-option 3 ใช้จุดเกาส์ 4 จุด

-หาก option เป็นค่าอื่น ๆ จะใช้จุดเกาส์ 1 จุด

1.เกาส์ควอดราเจออร์ของเอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรง

```
1  import numpy as np
2   def gaussQuadratureT3(option):
3    if option == 1: # 3 จุดเกาส์ option 1
4       locations = np.array([[0.5, 0.5],
5                             [0, 0.5],
6                             [0.5, 0]])
7       weights = 1/6*np.ones((3,1))
8   elif option == 2: # 3 จุดเกาส์ option 2
9       locations = np.array([[1/6, 1/6],
10                            [2/3, 1/6],
11                            [1/6, 2/3]])
12        weights = 1/6*np.ones((3,1))
13    elif option == 3: # 4 จุดเกาส์
14        locations = np.array([[1/3, 1/3],
15                            [1/5, 1/5],
16                            [3/5, 1/5],
17                            [1/5, 3/5]])
18        weights = np.array([-27/96, 25/96, 25/96
19                            ,25/96])
20    else:
21        locations = np.array([[1/3, 1/3]])
22        weights = 0.5*np.ones((1,1))
23    return weights,locations
```

คำสั่งที่ 3.5 เกาส์ควอดราเจออร์ของเอลิเมนต์รูปสามเหลี่ยมเชิงเส้นตรง

m	n_p	p	ξ_p	η_p	W_p	Position of points
1	1	1	1 / 3	1 / 3	1 / 2	
2	3	1	1 / 2	1 / 2	1 / 6	
		2	0	1 / 2	1 / 6	
		3	1 / 2	0	1 / 6	
2	3	1	1 / 6	1 / 6	1 / 6	
		2	2 / 3	1 / 6	1 / 6	
		3	1 / 6	2 / 3	1 / 6	
3	4	1	1 / 3	1 / 3	-27 / 96	
		2	1 / 5	1 / 5	25 / 96	
		3	3 / 5	1 / 5	25 / 96	
		4	1 / 5	3 / 5	25 / 96	

รูปที่ 3.2 ตารางเทียบเกาส์ควอดราเจออร์สำหรับเอลิเมนต์สามเหลี่ยม

2.เกาส์ควอดราเจออร์ของเอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่

```
1  import numpy as np
2   def gaussQuadrature(option):
3    if option == 1: # full integration
4        locations = np.array(
5                    [[-0.577350269189626,
6                       -0.577350269189626],
7                     [0.577350269189626,
8                       -0.577350269189626],
9                     [0.577350269189626,
10                       0.577350269189626],
11                    [-0.577350269189626,
12                       0.577350269189626]])
13        weights = np.ones((4,1))
14    else:
15        locations =np.zeros((1,2))
16        weights = 4
17    return weights,locations
```

คำสั่งที่ 3.6 เกาส์ควอดราเจออร์ของเอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่

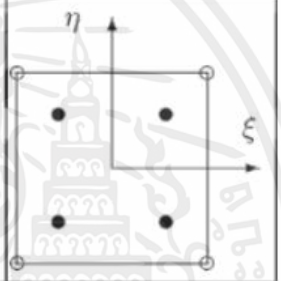
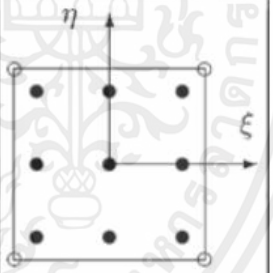
3.เกาส์ควอดราเจออร์ของเอลิเมนต์รูปสี่เหลี่ยมกำลังสอง

```
import numpy as np
 def gaussQuadratureQ8(option):
  if option == 2:
   # full 2nd order integration: 3x3=9 จุดเกาส์
      locations = np.array([[-np.sqrt(3/5),
                             -np.sqrt(3/5)],
                            [0, -np.sqrt(3/5)],
                            [np.sqrt(3/5),
                             -np.sqrt(3/5)],
                            [-np.sqrt(3/5),0],
                            [0, 0],
                            [np.sqrt(3/5),0],
                            [-np.sqrt(3/5),
                              np.sqrt(3/5)],
                           [0, np.sqrt(3/5)],
                           [np.sqrt(3/5),
                            np.sqrt(3/5)]])
      weights =  np.array([[25/81],[40/81],[25/81],
    [40/81],[64/81],[40/81],[25/81],[40/81],[25/81]])

  elif option == 1:
    # (linear)integration: 2x2=4  จุดเกาส์
      locations = np.array([[-0.577350269189626,
                              -0.577350269189626],
                             [0.577350269189626,
                              -0.577350269189626],
                             [0.577350269189626,
                               0.577350269189626],
                             [-0.577350269189626,
                               0.577350269189626]])
      weights = np.ones((4,1))

```

```
33      else: # 1x1=1: 1 จุดเกาส์
34          locations =np.zeros((1,2))
35          weights = 4*np.ones((1,1))
36
37      return weights,locations
```

คำสั่งที่ 3.7 เกาส์ควอดราเจออร์ของเอลิเมนต์รูปสี่เหลี่ยมกำลังสอง

จากชุดคำสั่งในรูปข้างต้นมาจากตารางเทียบเกาส์ควอดราเจออร์

m	n_p	p	ξ_p	η_p	W_p	Position of points
1	1	1	0	0	4	
3	4	1	$-1/\sqrt{3}$	$-1/\sqrt{3}$	1	
		2	$+1/\sqrt{3}$	$-1/\sqrt{3}$	1	
		3	$-1/\sqrt{3}$	$+1/\sqrt{3}$	1	
		4	$+1/\sqrt{3}$	$+1/\sqrt{3}$	1	
5	9	1	$-\sqrt{3/5}$	$-\sqrt{3/5}$	$25/81$	
		2	0	$-\sqrt{3/5}$	$40/81$	
		3	$+\sqrt{3/5}$	$-\sqrt{3/5}$	$25/81$	
		4	$-\sqrt{3/5}$	0	$40/81$	
		5	0	0	$64/81$	
		6	$+\sqrt{3/5}$	0	$40/81$	
		7	$-\sqrt{3/5}$	$+\sqrt{3/5}$	$25/81$	
		8	0	$+\sqrt{3/5}$	$40/81$	
		9	$+\sqrt{3/5}$	$+\sqrt{3/5}$	$25/81$	

รูปที่ 3.3 ตารางเทียบเกาส์ควอดราเจออร์สำหรับเอลิเมต์สี่เหลี่ยม

3.2.5 ฟังก์ชันหาเมทริกซ์ความแข็ง (Stiffness Matrix)

ฟังก์ชัน formStiffness2D ใช้ในการคำนวณเมทริกซ์ความต้านทาน (stiffness matrix) สำหรับปัญหาสองมิติ โดยใช้วิธีการ Quadrature (Gauss Quadrature) ในการประมาณค่าอินทิกรัลของการคำนวณ พารามิเตอร์ที่ใช้ในการคำนวณ

nDof : จำนวนของ Degree of Freedom หรือตำแหน่งของโหนด

nE : จำนวนของอิลิเมนต์

eNodes : เมทริกซ์ที่บ่งชี้โหนดของแต่ละอิลิเมนต์

nP : จำนวนของโหนดต่อแต่ละอิลิเมนต์

xy : เมทริกซ์ที่เก็บพิกัดของโหนด

C : เมทริกซ์ค่าคงที่ในสมการของความต้านทาน

h : ความหนาของอิลิเมนต์

หลังจากฟังก์ชันได้รับค่าอินพุตมาแล้วทำการสร้างเมทริกซ์ K ที่มีขนาด (nDof, nDof) และเติมค่าศูนย์เข้าไปในเมทริกซ์นี้ จากนั้นใช้ฟังก์ชัน gaussQuadrature เพื่อรับค่าของค่าน้ำหนักและตำแหน่งของจุด Gauss Quadrature ที่ใช้ในการประมาณค่าอินทิกรัล จากนั้นทำการวนลูปเอลิเมนต์เพื่อหาค่าเมทริกซ์ความต้านทานของแต่ละเอลิเมนต์

กำหนดตำแหน่งของ Degree of freedom ในแต่ละเอลิเมนต์ลงในเมทริกซ์ eDof จากนั้นคำนวณค่าเมทริกซ์ B โดยใช้ค่าตำแหน่งของโหนดและอินทิกรัลของฟังก์ชันรูปร่าง นำเมทริกซ์ B ไปคำนวณหาค่าเมทริกซ์ความต้านทานของแต่ละเอลิเมนต์ (K^e) โดยใช้เกาส์พอยต์ตามประเภทเอลิเมนต์ที่ใช้ในการแก้ปัญหา แล้วนำค่าเมทริกซ์ความต้านทานของแต่ละเอลิเมนต์ไปเพิ่มค่าในเมทริกซ์ความต้านทานรวม K ตามตำแหน่ง Degree of freedom นั้นเมื่อวนลูปครบทุกตำแหน่งทำการคืนค่า K

```
1  import numpy as np
2   def formStiffness2D(nDof,nE,eNodes,nP,xy,C,h):
3    K = np.zeros((nDof,nDof))
4    gaussWt, gaussLoc = gaussQuadrature(1)
5    for e in range(nE):
6        id = eNodes[e,:]
7        eDof = np.zeros((8,1))
8        eDof[0:4,0] = id
9        eDof[4:8,0] = id + nP
10        eDof = eDof.flatten()
11
```

```
12          ndof = id.size
13         # คิดแต่ละจุดเกาส์
14          for q in range(gaussWt.size):
15
16              GaussPoint = gaussLoc[q,:]
17              xi = GaussPoint[0]
18              eta = GaussPoint[1]
19
20         # หาฟังก์ชันประมาณค่าและหาค่าอนุพันธ์ของเอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่
21              shape,nDeriv = shapeFuncQ4(xi,eta)
22
23         # หาเมทริกซ์จาโคเบียนและอินเวิร์ดเมทริกจาโคเบียน
24              J,xyDeriv = Jacobian(xy[id-1,:],nDeriv)
25
26         # หาเมทริกซ์ B (Linear strain - displacement matrix)
27              B = np.zeros((3,2*ndof))
28              B[0,0:ndof]       =
29                                np.transpose(xyDeriv[:,0])
30              B[1,ndof:(2*ndof)]=
31                                np.transpose(xyDeriv[:,1])
32              B[2,0:ndof]       =
33                                np.transpose(xyDeriv[:,1])
34              B[2,ndof:(2*ndof)]=
35                                np.transpose(xyDeriv[:,0])
36
37         # หาเมทริกซ์ความต้านทาน (stiffness matrix)
38              BT = np.transpose(B)
39              detJ = np.linalg.det(J)
40              Ke =
41            np.matmul(np.matmul(BT,C),B)*h*detJ*gaussWt[q]
42              for ii in range(np.size(Ke,0)):
43                 row = int(eDof[ii])-1
44                  for jj in range(np.size(Ke,1)):
```

```
45                      col = int(eDof[jj])-1
46                      K[row,col] = K[row,col] + Ke[ii,jj]
47     return K
```

คำสั่งที่ 3.6 ฟังก์ชันหาเมทริกซ์ความแข็งของเอลิเมนต์รูปสี่เหลี่ยมเชิงเส้นคู่

```
1   import numpy as np
2    def formStiffness2DQ8(nDof,nE,eNodes,nP,
3                          xy,C,h,gauss_quad_opt):
4    K = np.zeros((nDof,nDof))
5    gaussWt, gaussLoc= gaussQuadratureQ8(gauss_quad_opt)
6    for e in range(nE):
7        id = eNodes[e,:]
8        eDof = np.zeros((16,1))
9        eDof[0:8,0] = id
10        eDof[8:16,0] = id + nP
11        eDof = eDof.flatten()
12
13        ndof = id.size
14       # คิดแต่ละจุดเกาส์
15        for q in range(gaussWt.size):
16
17            GaussPoint = gaussLoc[q,:]
18            xi = GaussPoint[0]
19            eta = GaussPoint[1]
20
21       # หาฟังก์ชันประมาณค่าและหาค่าอนุพันธ์ของเอลิเมนต์รูปสี่เหลี่ยมกำลังสอง
22            shape,nDeriv = shapeFuncQ8(xi,eta)
23
24       # หาเมทริกซ์จาโคเบียนและอินเวิร์ดเมทริกจาโคเบียน
25            J,xyDeriv = Jacobian(xy[id-1,:],nDeriv)
26
27       # หาเมทริกซ์ B (Linear strain - displacement matrix)
28            B = np.zeros((3,2*ndof))
29            B[0,0:ndof]      =
```

```
                            np.transpose(xyDeriv[:,0])
            B[1,ndof:(2*ndof)] =
                            np.transpose(xyDeriv[:,1])
            B[2,0:ndof]       =
                            np.transpose(xyDeriv[:,1])
            B[2,ndof:(2*ndof)] =
                            np.transpose(xyDeriv[:,0])
            # stiffness matrix
            BT = np.transpose(B)
            detJ = np.linalg.det(J)
            Ke =
          np.matmul(np.matmul(BT,C),B)*h*detJ*gaussWt[q]
            for ii in range(np.size(Ke,0)):
                row = int(eDof[ii])-1
                for jj in range(np.size(Ke,1)):
                    col = int(eDof[jj])-1
                    K[row,col] = K[row,col] + Ke[ii,jj]
    return K
```

คำสั่งที่ 3.7 ฟังก์ชันหาเมทริกซ์ความแข็งของเอลิเมนต์รูปสี่เหลี่ยมกำลังสอง

```
import numpy as np
def formStiffness2DT3(nDof,nE,eNodes,nP,xy,
                      C,h,gauss_quad_opt):
 K = np.zeros((nDof,nDof))
 gaussWt, gaussLoc =
                    gaussQuadratureT3(gauss_quad_opt)
 for e in range(nE):
     id = eNodes[e,:]
     eDof = np.zeros((6,1))
      eDof[0:3,0] = id
      eDof[3:6,0] = id + nP
      eDof = eDof.flatten()

```

```
14          ndof = id.size # จำนวนโหนดแต่ละเอลิเมนต์
15
16          # คิดแต่ละจุดเกาส์
17          for q in range(gaussWt.size):
18
19           GaussPoint = gaussLoc[q,:]
20              xi = GaussPoint[0]
21              eta = GaussPoint[1]
22
23         # หาฟังก์ชันประมาณค่าและหาค่าอนุพันธ์ของเอลิเมนต์สามเหลี่ยมเชิงเส้นตรง
24              shape,nDeriv = shapeFuncT3(xi,eta)
25
26         # หาเมทริกซ์จาโคเบียนและอินเวิร์ดเมทริกจาโคเบียน
27              J,xyDeriv = Jacobian(xy[id-1,:],nDeriv)
28
29         # หาเมทริกซ์B (Linear strain - displacement matrix)
30              B = np.zeros((3,2*ndof))
31           B[0,0:ndof]      =
32                                np.transpose(xyDeriv[:,0])
33            B[1,ndof:(2*ndof)] =
34                                 np.transpose(xyDeriv[:,1])
35            B[2,0:ndof]      =
36                                np.transpose(xyDeriv[:,1])
37            B[2,ndof:(2*ndof)] =
38                                np.transpose(xyDeriv[:,0])
39
40              # stiffness matrix
41              BT = np.transpose(B)
42              detJ = np.linalg.det(J)
43              Ke =
44           np.matmul(np.matmul(BT,C),B)*h*detJ*gaussWt[q]
45              for ii in range(np.size(Ke,0)):
46                  row = int(eDof[ii])-1
```

```
47                 for jj in range(np.size(Ke,1)):
48                     col = int(eDof[jj])-1
49                     K[row,col] = K[row,col] + Ke[ii,jj]
50     return K
```

คำสั่งที่ 3.8 ฟังก์ชันหาเมทริกซ์ความแข็งของรูปสามเหลี่ยมเชิงเส้นตรง

3.3 ชุดคำสั่งหลัก (Main program)

3.3.1 การใส่ค่าตัวแปรต่างๆ (Input Parameters)

เป็นการกำหนดรูปร่าง, ขนาด และสมบัติของวัสดุ ซึ่งมีการใส่ค่าตัวแปรต่างๆ แตกต่างกันตามรูปร่างของชิ้นงาน

```
# คุณสมบัติวัสดุ (Material Pros)
E = ***
h = ***
pois = ***
Lx = ***
Ly = ***
Nx = ***
Ny = ***
dx = Lx/Nx
dy = Ly/Ny
# หาเมทริกซ์C (Stress- strain matrix: C - Plane stress)
E1 =E/(1-pois**2)
C = E1*np.array([[1, pois, 0],[pois, 1, 0],[0, 0,
(1-pois)/2]])

# ค่า tolerance ที่จะใช้ในชุดคำสั่ง
tol = ***
```

คำสั่งที่ 3.9 การใส่ค่าตัวแปรต่าง

เป็นการใส่ข้อมูลค่าที่จำเป็นในการคำนวณ โดยมีรายการดังนี้

E	คือ มอดุลัสของยัง (Young's modulus)
h	คือ ค่าความหนา (เมตร)
pois	คือ อัตตราส่วนของปัวซอง (Poisson's ratio)
Lx	คือ ความยาวตามแนวแกน x (เมตร)
Ly	คือ ความยาวตามแนวแกน y (เมตร)
tol	คือ ค่าที่มีความเล็กมากๆมีจุดประสงค์เพื่อใช้ในการตรวจสอบเงื่อนไขบางอย่าง
C	คือ เมทริกซ์ความสัมพันธ์ระหว่างความเค้นและความเครียด (Stress-Strain Matrix)

3.3.2 การนำเข้าตาข่าย

ในกรณีที่กำหนดตำแหน่งโหนดและสร้างตาข่ายของชิ้นงานรูปสี่เหลี่ยม สามารถแสดงตาข่ายสำหรับรูปสี่เหลี่ยมได้ด้วยคำสั่งต่อไปนี้

```
# สร้างโหนด
x = np.linspace(0, Lx, Nx+1)
y = np.linspace(0, Ly, Ny+1)
xx, yy = np.meshgrid(x, y)
xy = np.column_stack((xx.flatten(), yy.flatten()))
# เพิ่มโหนดในแต่ละเอลิเมนต์
nNodes = (Nx+1)*(Ny+1)
nElems = Nx*Ny
eNodes = np.zeros((nElems, 4), dtype=int)
for j in range(Ny):
    for i in range(Nx):
        n1 = j*(Nx+1) + i
        n2 = n1 + 1
        n3 = n2 + Nx + 1
        n4 = n3 - 1
        e = j*Nx + i
        eNodes[e,:] = [n1, n2, n3, n4]
nE=len(eNodes)
nP=len(xy)
nDof=2*nP
xx.flatten()
yy.flatten()
nnel = np.size(eNodes,1)
for iel in range(nE):
    for i in range(nnel):
        eNodes[iel,i] = 1 + eNodes[iel,i]
```

คำสั่งที่ 3.10 ฟังก์ชันการสร้างตาข่ายสำหรับสี่เหลี่ยม

ข้อมูลที่นำเข้าที่ต้องการเพื่อประกอบเป็นตาข่าย และใช้ในการคำนวณมีอยู่สองส่วนคือ ตำแหน่งของแต่ละโหนด (Nodal Coordinates) และโหนดของแต่ละเอลิเมนต์ (Nodal Connectivities) โดยเมื่อนำเข้าข้อมูลดังกล่าวแล้วก็ทำการตรวจสอบว่าค่าที่เรานำเข้านั้นถูกต้องตามที่ต้องการหรือไม่ ในกรณีที่ชิ้นงานมีความซับซ้อนสามารถนำเข้าตำแหน่งโหนดและโหนดในแต่ละเอลิเมนต์จากภายนอกสามารถนำเข้าได้ด้วยคำสั่งต่อไปนี้

```
1  dfnode = pd.read_csv('NLIST4n4e.csv', header=None)
2  dfnode.head()
3  dfnode = dfnode.dropna(axis=0)
4  dfnode.head()
5  dfnode1 = dfnode[dfnode[0].isin(["Node Number"])]
6  dfnode = pd.concat([dfnode, dfnode1,
                       dfnode1]).drop_duplicates(keep=False)
8  dfnode.head()
9  nP = dfnode.count(axis=0)
10 nP = nP[0]
11 nDof = 2*nP
12 xy = dfnode.iloc[:,1:3]
13 xy = xy.reset_index()
14 xy = xy.iloc[:,1:3]
15 xy = xy.to_numpy(dtype=np.float32)
16 dfeNode = pd.read_csv('ELIST4n4e.csv', header=None)
17 dfeNode = dfeNode.dropna(axis=0)
18 dfeNode = dfeNode[[2,3,4,5]]
19 eNodes = dfeNode[[2,3,4,5]]
20 nE = eNodes.count(axis=0)
21 nE = nE[2]
22 eNodes.head()
23 eNodes = eNodes.astype(int)
24 eNodes = eNodes.to_numpy(dtype=np.int32)
25 eNodes.shape
26 PlotMesh(xy,eNodes,nE)
```

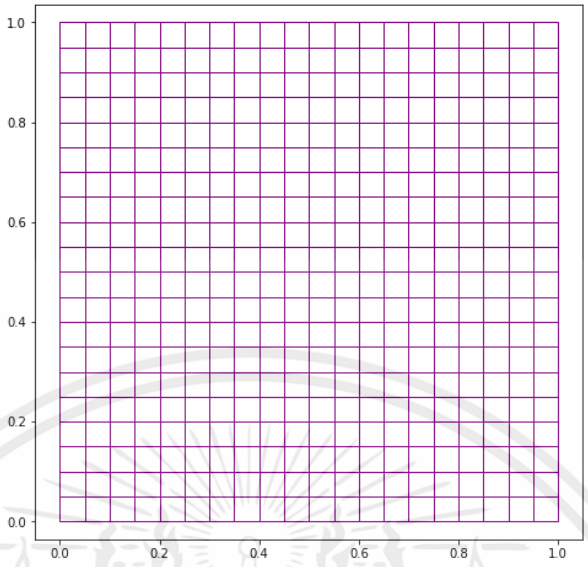
คำสั่งที่ 3.12 ตัวอย่างการนำเข้าข้อมูลตำแหน่งโหนดในการสร้างตาข่าย

จากคำสั่งข้างต้นเป็นตัวอย่างที่ใช้นำข้อมูลตำแหน่งโหนดจากไฟล์ NLIST.csv และข้อมูลโหนดในแต่ละเอลิเมนต์จากไฟล์ ELIST1.csv มาสร้างตาข่ายด้วยฟังก์ชันแสดงตาข่ายโดยสามารถแสดงตาข่ายที่สร้างขึ้นได้โดยใช้ชุดคำสั่ง

```
1  # นำค่า xy และ eNodes มาแสดงตาข่าย
2  def PlotMesh(xy,eNodes,nel):
3      nnel = np.size(eNodes,1)
4      X = np.zeros((nnel,nel))
5      Y = np.zeros((nnel,nel))
6
7      for iel in range(nel):
8          for i in range(nnel):
9              ndi = eNodes[iel,i]
10             X[i,iel] = xy[ndi-1,0]
11             Y[i,iel] = xy[ndi-1,1]
12
13     plt.figure(figsize=(8, 8))
14     plt.axis('equal')
15     plt.fill(X, Y, facecolor='none',
16              edgecolor='purple',   linewidth=1)
17     plt.show()
```

คำสั่งที่ 3.11 ฟังก์ชันการแสดงตาข่าย

โดยมีตัวอย่างผลลัพธ์จากฟังก์ชันการแสดงตาข่ายดังนี้



รูปที่ 3.5 ตัวอย่างผลลัพธ์จากฟังก์ชันการสร้างตาข่าย

3.3.3 การคำนวณเมทริกซ์ความแข็ง

ใช้คำสั่งที่ 3.13 เพื่อหาเมทริกซ์ K

```
1 K = formStiffness2D(nDof,nE,eNodes,nP,xy,C,h)
```

คำสั่งที่ 3.13 การหาเมทริกซ์ความแข็งจากฟังก์ชันหาเมทริกซ์ความแข็ง

3.3.4 เงื่อนไขที่ขอบ (Boundary Condition)

กำหนดเงื่อนไขที่ขอบ คือ จุดที่ไม่ขยับหรือถูกยึดอยู่กับที่ และสร้างเมทริกซ์ fixDof จากเงื่อนไขนั้น ยกตัวอย่างเช่น แผ่นเรียบที่ถูกยึดไว้กับที่บริเวณพิกัด x เท่ากับ 0 สามารถแสดงได้ดังนี้

```
1 fixP = np.argwhere(xy[:,0]<=tol)
2 fixDof = np.array([fixP,fixP+nP],order = 'F')
3 fixDof = fixDof.flatten()
4 fixDof
```

คำสั่งที่ 3.14 ตัวอย่างการกำหนดเงื่อนไขที่ขอบ

3.3.5 การใส่ภาระ (Loading)

ในที่นี้เราพิจารณาความหยืดหยุ่นในสองมิติ ดังนั้นภาระที่เราใส่คือแรงที่มากระทำ โดยเขียนคำสั่งให้สร้างเวกเตอร์แรงที่มากระทำขึ้นมา โดยค่าภายในเมทริกซ์ขึ้นอยู่กับตำแหน่งที่แรงมากระทำและขนาดของแรง สามารถใช้ฟังก์ชันเหล่านี้ในการสร้างเวกเตอร์แรงที่มากระทำ

```
# non-uniform force, triangualar distribution
bottomY_dfx = ***
topY_dfx = ***
#function dfx respect to y

def nonuniforce(y): # non-uniform force, triangualar
distribution

dfx = (((topY_dfx-bottomY_dfx)/Ly)*y)+bottomY_dfx

return dfx

force = np.zeros((nDof,1))
loadP1 = np.where((xy[:,0]>=Lx-tol) & (xy[:,1]>=Ly-
tol))
loadP2 = np.where((xy[:,0]>=Lx-tol) & (xy[:,1]<=tol))
loadP3 = np.where((xy[:,0]>=Lx-tol) & (xy[:,1]>=tol)
&
(xy[:,1]<=Ly-tol))

#force function
def Forcetype(option):
  if option == 1: # uniform square
      dfx = bottomY_dfx
      force[loadP1,0] = dfx*dy/2
      force[loadP2,0] = dfx*dy/2
      force[loadP3,0] = dfx*dy

  else:
```

```
28        force[loadP1,0] = nonuniforce(xy[loadP1,1])*dy/2
29        force[loadP2,0] = nonuniforce(xy[loadP2,1])*dy/2
30        force[loadP3,0] = nonuniforce(xy[loadP3,1])*dy
31
32    return force
33
34 Forcetype(1)
```

คำสั่งที่ 3.15 การสร้างเวกเตอร์ของแรง

จากคำสั่งที่ 3.15 พบว่ามีการแบ่งขนาดของแรงสามตำแหน่ง เนื่องจากโหนดที่บริเวณขอบบนและขอบล่างของแผ่นเรียบมีพื้นที่ในการรับแรงที่มากระทำเพียงครึ่งเดียวจากโหนดอื่นๆที่รับแรงที่มากระทำ

3.2.6 หาผลเฉลยของสมการ (Solve EOM)

ใช้ฟังก์ชันผลเฉลยที่สร้างไว้แล้วเก็บค่า disp ที่ได้ ซึ่ง disp เป็นผลเฉลยของสมการและปัญหาที่เราสนใจหรือกำหนดขึ้นมา และนำไปใช้ต่อในขั้นตอนหลังการประมวลผลอีก

```
1 disp = solution(nDof,fixDof,K,force)
2 dispu = disp[0:nP,0]
3 dispv = disp[nP:2*nP,0]
```

คำสั่งที่ 3.16 การหาผลเฉลยโดยฟังก์ขันผลเฉลย

จากรูปที่ หลังจากได้เวกเตอร์ของการเสียรูปในรูปของตัวแปร disp ให้ทำการแยกเวกเตอร์การเสียรูป u และ v ให้อยู่ในตัวแปร dispu และ dispv ตามลำดับ

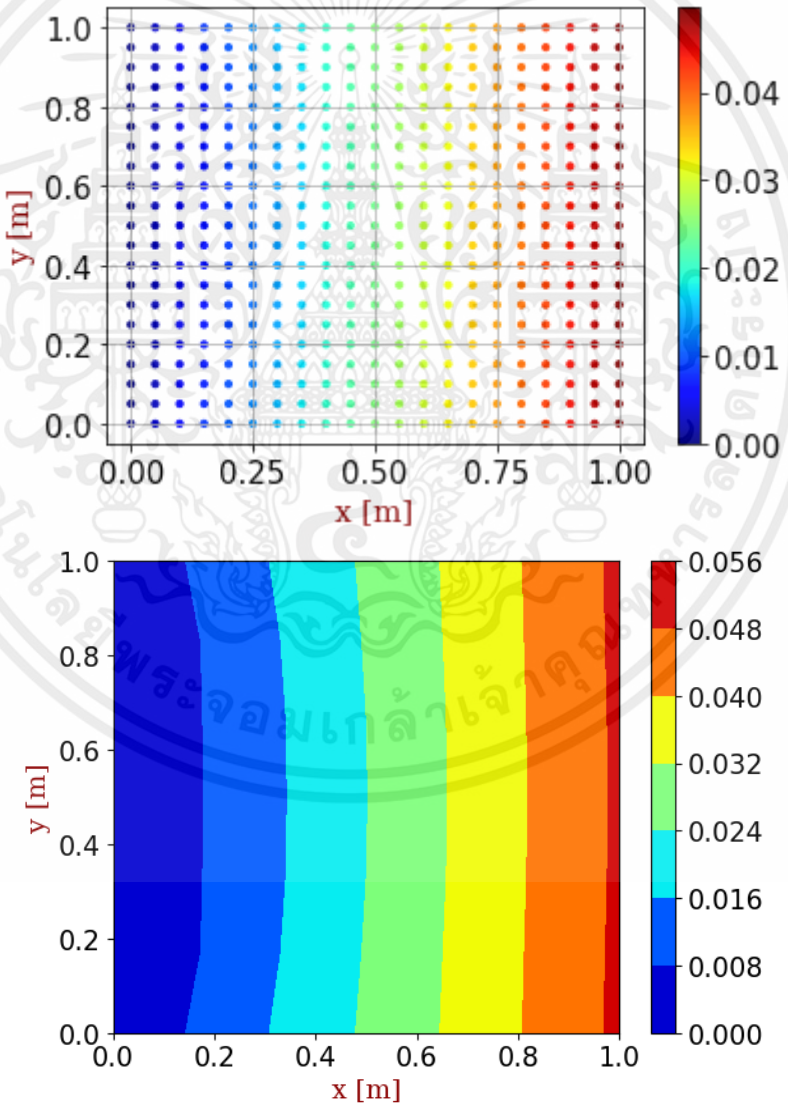
ซึ่งสามารถนำตัวแปร dispu มาแสดงผลได้โดยคำสั่งต่อไปนี้

```
1 ## plot displacement U
2 plt.rcParams['font.size'] = 15
3 ax = plt.axes()
4 plt.scatter(xy[:,0], xy[:,1], s=10, c=dispu)
5 plt.grid()
6  font1 = {'family':'serif','color':'darkred',
        'size':15}
7  plt.xlabel("x [m]", fontdict = font1)
8  plt.ylabel("y [m]", fontdict = font1)
9  plt.set_cmap('jet')
10 plt.colorbar()
11 plt.savefig('dispU', bbox_inches='tight',
              dpi=300)
```

```
1  ##plot displacement U contour
2  plt.rcParams['font.size'] = 15
3  fig, ax = plt.subplots()
4  cf = ax.contourf(x, y, dispu.reshape((len(x),
```

```
                    len(y))))
fig.colorbar(cf)

ax.set_xlabel("x [m]", fontdict={'family': 'serif',
                'color': 'darkred', 'size': 15})
ax.set_ylabel("y [m]", fontdict={'family': 'serif',
                'color': 'darkred', 'size': 15})
plt.savefig('dispU_contourf', bbox_inches='tight',
            dpi=300)
```

คำสั่งที่ 3.17,3.18 คำสั่งแสดงผลการเสียรูป น



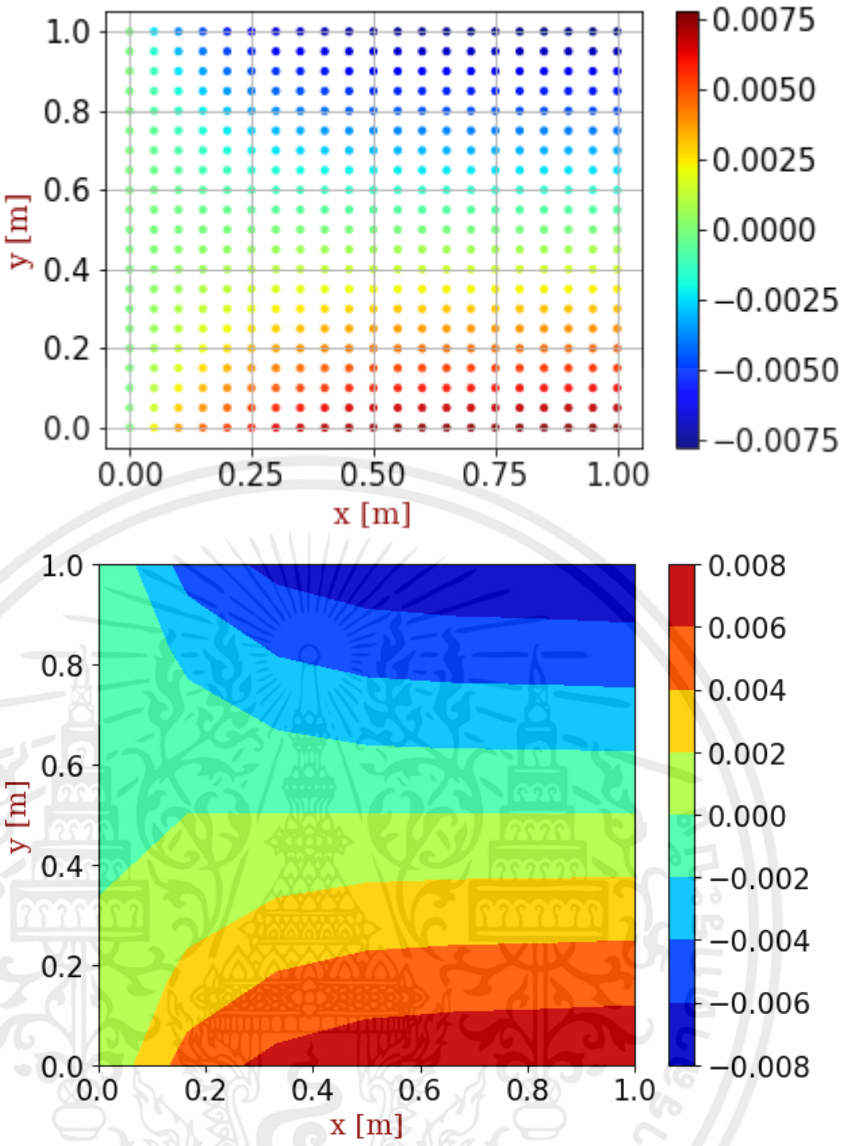
รูปที่ 3.8,3.9 ตัวอย่างการแสดงผลการเสียรูป น

ในส่วนตัวแปร dispv สามารถแสดงผลได้โดยคำสั่งแบบเดียวกันดังนี้

```
## plot displacement V
plt.rcParams['font.size'] = 15
ax = plt.axes()
plt.scatter(xy[:,0], xy[:,1], s=10, c=dispv)
plt.grid()
font1 = {'family':'serif','color':'darkred',
         'size':15}
plt.xlabel("x [m]", fontdict = font1)
plt.ylabel("y [m]", fontdict = font1)
plt.set_cmap('jet')
plt.colorbar()
plt.savefig('dispV', bbox_inches='tight', dpi=300)
```

```
##plot displacement V contour
plt.rcParams['font.size'] = 15
fig, ax = plt.subplots()
cf = ax.contourf(x, y, dispv.reshape((len(x),
                 len(y))))
fig.colorbar(cf)

ax.set_xlabel("x [m]", fontdict={'family': 'serif',
              'color': 'darkred', 'size': 15})
ax.set_ylabel("y [m]", fontdict={'family': 'serif',
               'color': 'darkred', 'size': 15})

plt.savefig('dispU_contourf', bbox_inches='tight',
            dpi=300)
```

คำสั่งที่ 3.19,3.20 คำสั่งแสดงผลการเสียรูป v



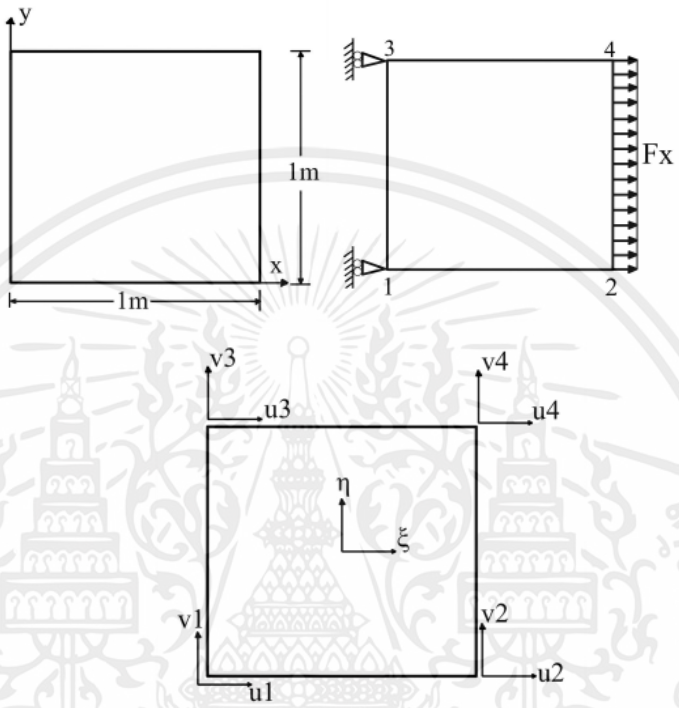
รูปที่ 3.10,3.11 ตัวอย่างการแสดงผลการเสียรูป v

3.4 หลังการประมวลผล (Post-processing)

นำผลเฉลยที่ได้มาสร้างกราฟแสดงผลเพื่อดูภาพรวมของการเปลี่ยนแปลงที่เกิดขึ้น หรือนำผลเฉลยที่ได้ไปหาความผิดพลาดเทียบกับซอฟต์แวร์อื่น เช่น Ansys เป็นต้น

บทที่ 4
ตัวอย่างการคำนวณด้วยชุดคำสั่งไพทอน

ตัวอย่างที่ใช้ในการเสนอผลลัพธ์ที่ได้จากชุดคำสั่งไพทอน คือ



รูปที่ 4.1 ภาพปัญหาที่ใช้ในการคำนวณ

โดยกำหนดคุณสมบัติของชิ้นงาน เงื่อนไขที่ขอบ และแรงดังนี้

คุณสมบัติของชิ้นงาน

E = 2×10^{11} $\mathrm{N/m^2}$

h = 0.01 m

pois = 0.3

Lx = 1.0 m

Ly = 1.0 m

เงื่อนไขที่ขอบ จุดที่ไม่มีการเปลี่ยนตำแหน่งหรือถูกยึดไว้กับที่ซึ่งดังรูปข้างต้นคือบริเวณพิกัดที่ x เท่ากับ 0 โดยยึดเฉพาะการเปลี่ยนตำแหน่งในแกน x เท่านั้น

แรง เป็นแรงสม่ำเสมอขนาด 10^8 N/m กระทำทางด้านขวามือของชิ้นงาน

4.1 การเตรียมข้อมูล (Pre-processing)

เมื่อใส่คุณสมบัติของชิ้นงานด้วยคำสั่งที่ 3.9 ทำให้ได้ เมทริกซ์ C คือ

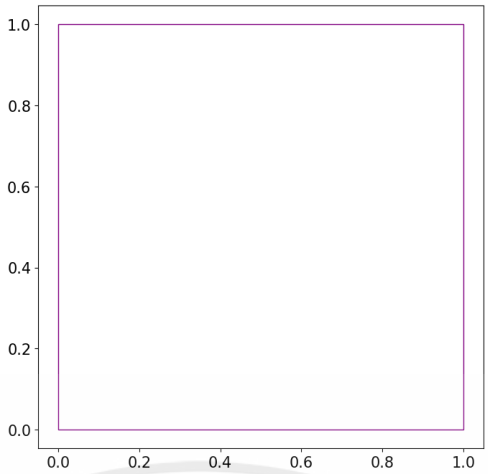
$$C = \begin{bmatrix} 2.1978E+11 & 65934065934 & 0 \\ 65934065934 & 2.1978E+11 & 0 \\ 0 & 0 & 76923076923 \end{bmatrix}$$

ทำการแบ่งชิ้นงานที่สนใจเป็นเอลิเมนต์ย่อยๆ หรือก็คือการสร้างตาข่ายโดยใช้คำสั่งที่ 3.10 ได้ตำแหน่งโหนดต่างๆดังนี้

Element	Node	(x,y)	Dof
1	1	0,0	u1
			v1
	2	1,0	u2
			v2
	4	1,1	u4
			v4
	3	0,1	u3
			v3

ตารางที่ 4.1 แสดงจำนวนเอเลเมนต์ จำนวนโหนด ตำแหน่งของโหนด และ ดีกรี ออฟ ฟรีดอม

โดยเมื่อนำค่าในตารางมาแสดงตาข่ายด้วยคำสั่งที่ 3.11 จะได้ตาข่ายดังแสดง



รูปที่ 4.2 ภาพแสดงตาข่ายที่ได้จากคำสั่งที่ 3.11

กำหนดแรงโดยคำสั่งที่ 3.15 ซึ่งทำให้ได้เมทริกซ์ของแรงคือ

$$F = \begin{bmatrix} 0 \\ 50000000 \\ 0 \\ 50000000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} u1 \\ u2 \\ u3 \\ u4 \\ v1 \\ v2 \\ v3 \\ v4 \end{matrix}$$

และกำหนดเงื่อนไขที่ขอบโดยคำสั่งที่ 3.14

$$fixDof = \begin{bmatrix} Dof \\ 0 \\ 2 \end{bmatrix}$$

4.2 การประมวลผล (Processing)

ใช้คำสั่ง **formStiffness2D(nDof,nE,eNodes,nP,xy,C,h):**ได้เมทริกซ์ความแข็งดังนี้

K =

u1	u2	u3	u4	v1	v2	v3	v4	
9.89E+08	-6E+08	1.1E+08	-4.9E+08	3.57E+08	-2.7E+07	27472527	-3.6E+08	u1
-6E+08	9.89E+08	-4.9E+08	1.1E+08	27472527	-3.6E+08	3.57E+08	-2.7E+07	u2
1.1E+08	-4.9E+08	9.89E+08	-6E+08	-2.7E+07	3.57E+08	-3.6E+08	27472527	u3
-4.9E+08	1.1E+08	-6E+08	9.89E+08	-3.6E+08	27472527	-2.7E+07	3.57E+08	u4
3.57E+08	27472527	-2.7E+07	-3.6E+08	9.89E+08	1.1E+08	-6E+08	-4.9E+08	v1
-2.7E+07	-3.6E+08	3.57E+08	27472527	1.1E+08	9.89E+08	-4.9E+08	-6E+08	v2
27472527	3.57E+08	-3.6E+08	-2.7E+07	-6E+08	-4.9E+08	9.89E+08	1.1E+08	v3
-3.6E+08	-2.7E+07	27472527	3.57E+08	-4.9E+08	-6E+08	1.1E+08	9.89E+08	v4

หลังจากได้เมทริกซ์ความแข็งของทั้งระบบแล้วใช้คำสั่ง **solution(nDof, fixDof, K, force):** เพื่อหาค่าการเปลี่ยนตำแหน่ง

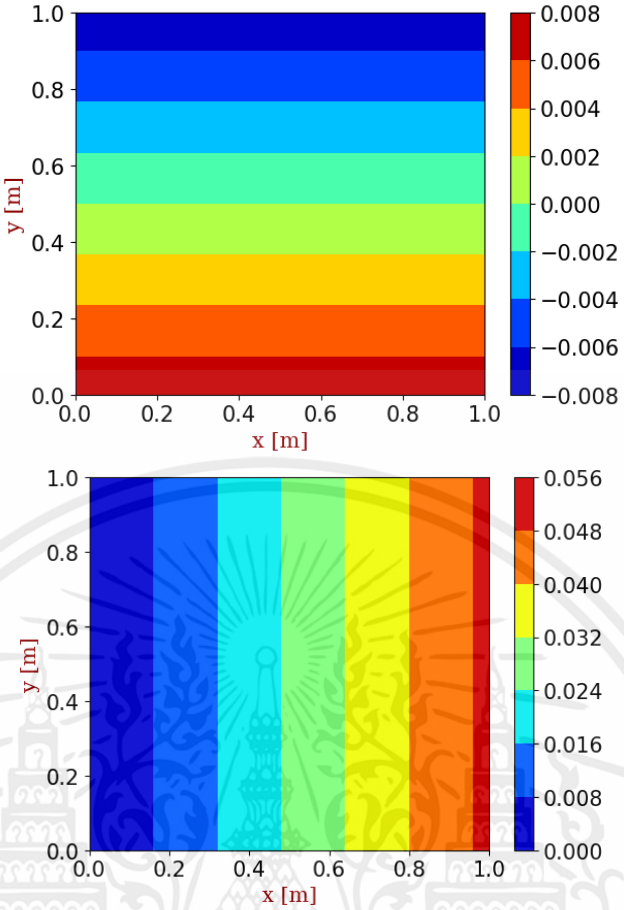
disp =

0	u1
0.05	u2
0	u3
0.05	u4
0.0075	v1
0.0075	v2
-0.0075	v3
-0.0075	v4

4.3 หลังการประมวลผล (Post-processing)

หลังจากขั้นตอน processing ที่ได้มาซึ่งคำตอบหรือก็คือค่าการเปลี่ยนตำแหน่ง disp มาใช้ในการคำนวณหรือแสดงผลต่อไป

นำค่าการเปลี่ยนตำแหน่งที่ได้จากกระบวนการ processing มาสร้างกราฟคอนทัวร์(Contour) ของการเปลี่ยนตำแหน่งในแกน x และแกน y ด้วยคำสั่งที่ 3.19



รูปที่ 4.3,4.4 ภาพแสดงคอนทัวที่ได้จากคำสั่งที่ 3.19

4.4 การเปรียบเทียบผล (Validation)

หลังจากได้ผลลัพธ์จากชุดคำสั่งไพทอนแล้ว ขั้นตอนที่สำคัญที่จำเป็นต้องทำในลำดับถัดมาคือการตรวจสอบความถูกต้องของผลลัพธ์ที่ได้ โดยในที่นี้จะใช้การคำนวณพื้นฐานและซอฟแวร์ Ansys และ Solid works มาเปรียบเทียบ โดยจะทำการเปรียบเทียบเฉพาะค่าการเปลี่ยนตำแหน่งเพียงอย่างเดียว และดูแนวโน้มของกราฟคอนทัวร์ที่ได้ในกระบวนการ post-processing

4.4.1 การคำนวณด้วยวิธีพื้นฐาน

ทำการหาค่าความเค้นตามแกน x ของวัสดุจาก

$$\sigma_x = \frac{P}{A}$$

โดยที่ $P = 1 \times 10^8\ N, A = 0.01\ m^2$

ดังนั้นจะได้

$$\sigma_x = \frac{10^8}{0.01} = 10^{10}\ \text{Pa}$$

สำหรับในแกน y และ z พบว่า $\sigma_y = \sigma_z = 0$

และจาก $E = 2 \times 10^{11}\ \text{Pa}, v = 0.30$

ดังนั้นสามารถหาค่าความเครียดในแต่ละแกนจาก

$$\varepsilon_x = \frac{1}{E}[\sigma_x - v(\sigma_y + \sigma_z)] = \frac{1}{2 \times 10^{11}}[10^{10} - 0.3(0+0)] = 0.05$$

$$\varepsilon_y = \frac{1}{E}[\sigma_y - v(\sigma_x + \sigma_z)] = \frac{1}{2 \times 10^{11}}[0 - 0.3(10^{10} + 0)] = -0.015$$

$$\varepsilon_z = \frac{1}{E}[\sigma_z - v(\sigma_y + \sigma_x)] = \frac{1}{2 \times 10^{11}}[0 - 0.3(0 + 10^{10})] = -0.015$$

หาระยะยืดตามแกน X และแกน Y ได้ดังนี้

$$\delta_x = \varepsilon_x L_x = 0.05 \times 1 = 0.05\ \text{m}$$

$$\delta_y = \varepsilon_y L_y = -0.015 \times 1 = -0.015\ \text{m}$$

กำหนดให้โหนดที่ 1 และ 4 ถูกยึดให้อยู่กับที่ในแนวแกน x ดังนั้น $u_1 = u_3 = 0\ \text{m}$ จากค่าที่คำนวณได้จะได้ค่า $u_2 = 0.05\ \text{m}, u_4 = 0.05\ \text{m}, v_1 = v_2 = 0.0075\ \text{m}, v_3 = v_4 = -0.0075\ \text{m}$

	0.000	u1
	0.050	u2
	0.000	u3
	0.050	u4
disp =	0.0075	v1
	0.0075	v2
	-0.0075	v3
	-0.0075	v4

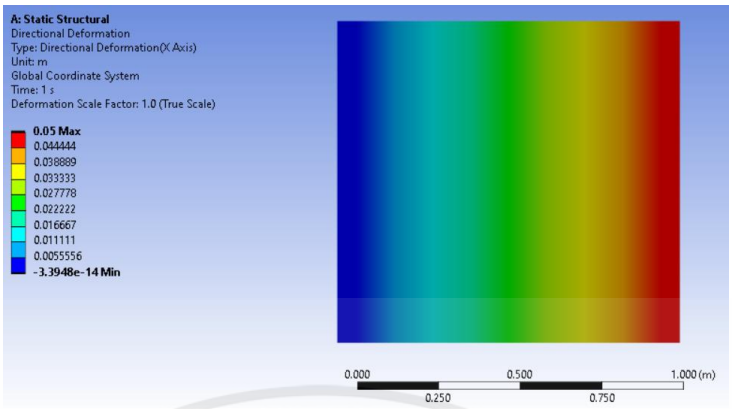
4.4.2 เปรียบเทียบค่าการเปลี่ยนตำแหน่ง

นำค่าการเปลี่ยนตำแหน่งจากแต่ละแหล่งมาใส่ลงในตารางเพื่อทำการเปรียบเทียบ ดังตารางต่อไปนี้

	Basic Cal.	Ansys	Solidworks	Python
u1	0	3.39E-14	0	0
u2	0.05	5.00E-02	5.29E-02	0.05
u3	0	3.39E-14	0	0
u3	0.05	5.00E-02	5.29E-02	0.05
v1	0.0075	7.50E-03	8.12E-03	0.0075
v2	0.0075	7.50E-03	8.12E-03	0.0075
v3	-0.0075	-7.50E-03	-7.23E-03	-0.0075
v4	-0.0075	-7.50E-03	-7.23E-03	-0.0075
	Avg. %error =	0.00%	6.67%	0.00%

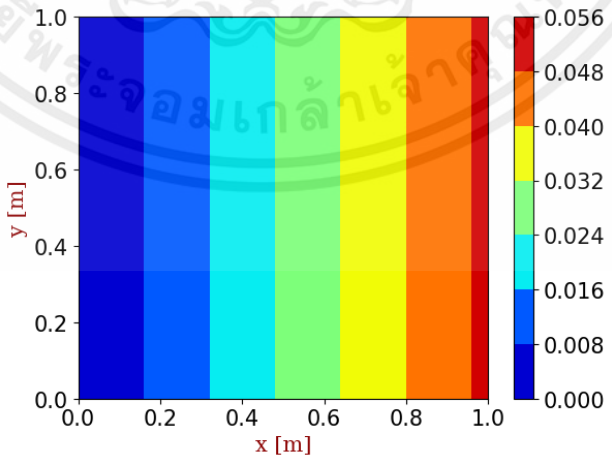
ตารางที่ 4.2 ตารางแสดงค่าเปรียบเทียบ

และเปรียบเทียบแนวโน้มของกราฟคอนทัวร์ ดังนี้

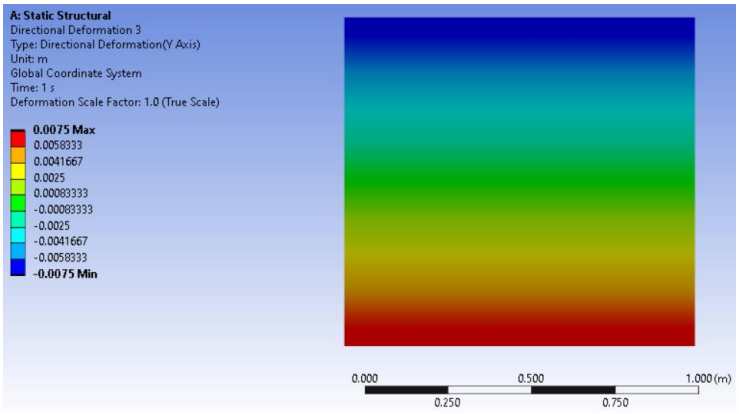


รูปที่ 4.4 ภาพแสดงคอนทัวแกน x ที่ได้จาก Ansys

รูปที่ 4.5 ภาพแสดงคอนทัวแกน x ที่ได้จาก Solid work

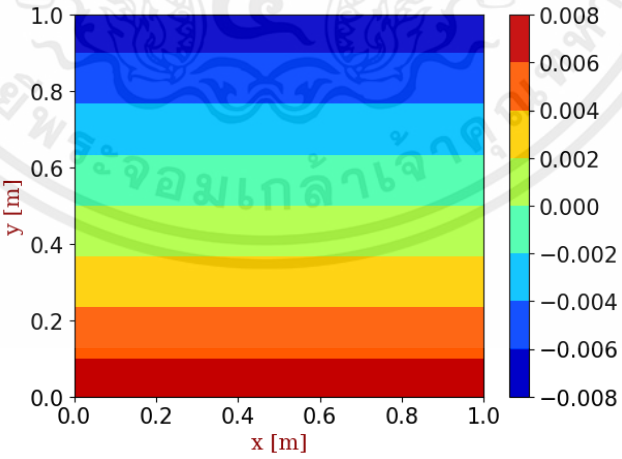


รูปที่ 4.6 ภาพแสดงคอนทัวแกน x ที่ได้จากคำสั่งที่ 3.19



รูปที่ 4.7 ภาพแสดงคอนทัวแกน y ที่ได้จาก Ansys

รูปที่ 4.8 ภาพแสดงคอนทัวแกน y ที่ได้จาก Solid work



รูปที่ 4.9 ภาพแสดงคอนทัวแกน y ที่ได้จากคำสั่งที่ 3.19

บทที่ 5
สรุปผลการดำเนินการ

วิทยานิพนธ์ฉบับนี้นำเสนอการออกแบบและพัฒนาชุดคำสั่งไพทอนที่ช่วยในการแก้ปัญหาเชิงวิศวกรรมเครื่องกลด้วยวิธีไฟไนต์เอลิเมนต์ โดยวัตถุประสงค์ของวิทยานิพนธ์ฉบับนี้คือ การศึกษาเกี่ยวกับวิธีการไฟไนต์เอลิเมนต์ เพื่อสร้างชุดคำสั่งไพทอนที่และจัดทำคู่มือในการทำความเข้าใจในวิธีการไฟไนต์เอลิเมนต์ และเพื่อสร้างชุดคำสั่งที่สามารถนำไปแก้ไขและพัฒนาต่อได้ในขอบเขตวิธีการไฟไนต์เอลิเมนต์

ขั้นตอนการพัฒนาชุดคำสั่งวิธีการไฟไนต์เอลิเมนต์ด้วยภาษาไพทอนนั้น แบ่งออกเป็นสามช่วงหลัก คือ ศึกษาและทำความเข้าใจวิธีการไฟไนต์เอลิเมนต์และภาษาคอมพิวเตอร์ไพทอน ทดลองเขียนชุดคำสั่งวิธีการไฟไนต์เอลิเมนต์ด้วยภาษาไพทอน ทดสอบและแก้ไขข้อผิดพลาดที่เกิดขึ้นในชุดคำสั่งรวมไปถึงผลลัพธ์ที่ได้จากชุดคำสั่ง โดยในขั้นตอนการสร้างตาข่ายของปัญหา การเลือกรูปร่างเอลิเมนต์สำหรับปัญหา และวิธีการแก้ระบบสมการเชิงเส้น ได้มีการเพิ่มทางเลือกในการใช้งานให้มากขึ้น เริ่มที่ขั้นตอนการสร้างตาข่ายของปัญหา สามารถทำได้อยู่สองวิธีในขณะนี้คือ การสร้างตาข่ายด้วยชุดคำสั่งไพทอนเอง และการนำเข้าข้อมูลตำแหน่งโหนดและรายการหมายเลขโหนดในแต่ละเอลิเมนต์จากซอฟแวร์อื่น โดยที่การสร้างตาข่ายด้วยชุดคำสั่งไพทอน สามารถสร้างตาข่ายได้เฉพาะปัญหาที่มีรูปร่างอย่างง่ายเช่น สี่เหลี่ยมและสามเหลี่ยม แต่การนำเข้าข้อมูลจากซอฟแวร์อื่นสามารถใช้ได้กับปัญหารูปแบบที่ซับซ้อนได้เท่าที่ซอฟแวร์นั้นสามารถทำได้ ในส่วนของวิธีการแก้ระบบสมการเชิงเส้นเพื่อหาผลเฉลย ได้นำวิธี Cholesky Decomposition และ Conjugate Gradient มาใช้ ซึ่งจากการทดสอบและศึกษาพบว่า วิธี Conjugate Gradient จะมีความเร็วในการคำนวณที่มากกว่าวิธี Cholesky Decomposition ในปัญหาที่มีจำนวนเอลิเมนต์มาก

ผลลัพธ์ของตัวอย่างปัญหาที่กำหนดขึ้นที่ได้จากชุดคำสั่งกับผลลัพธ์ที่ได้จากซอฟแวร์ Ansys มีความใกล้เคียงกับการคำนวณที่กล่าวถึงก่อนหน้านี้ แต่ทว่าความแตกต่างของผลลัพธ์จากซอฟแวร์ Solidworks เกิดจากข้อจำกัดในการกำหนดรูปร่างของเอลิเมนต์ที่ในซอฟแวร์ Solidworks ถูกกำหนดให้มีรูปร่างเป็นเอลิเมนต์รูปสามเหลี่ยม ซึ่งแตกต่างจากที่โจทย์กำหนดทำให้ผลลัพธ์มีความแตกต่างจากวิธีอื่น

ในปัจจุบัน ชุดคำสั่งวิธีการวิธีการไฟไนต์เอลิเมนต์ที่ได้ออกแบบและพัฒนาขึ้นสามารถแก้ปัญหากลศาสตร์ของแข็งในสภาพความยืดหยุ่นในสองมิติได้เป็นอย่างดี โดยที่ในชุดคำสั่งมีกำกับวิธีการใช้งานซึ่งช่วยให้ผู้ที่กำลังศึกษาเกี่ยวกับวิธีการไฟไนต์เอลิเมนต์ทำความเข้าใจได้มากขึ้น และเนื่องจากรูปแบบของชุดคำสั่งที่ได้พัฒนาขึ้นมานั้นเป็นชุดคำสั่งพื้นฐานที่ประกอบไปด้วยฟังก์ชันต่างๆ ที่จำเป็นสำหรับวิธีไฟไนต์เอลิเมนต์จึงทำให้สามารถนำไปแก้ไขและต่อยอดได้ ซึ่งนอกจากวิธีหรือคำสั่งที่ได้กล่าวไปแล้วข้างต้น ยังสามารถเพิ่มเติมวิธีหรือคำสั่งที่ช่วยเพิ่มประสิทธิภาพของชุดคำสั่งนี้ได้ไม่ว่าจะเป็น การศึกษาและทดลองเขียนคำสั่งในการกำกับหมายเลขโหนดใหม่ให้มีประสิทธิภาพเช่น การเรียงหมายเลขโหนดในแนวด้านที่สั้นที่สุดของตาข่ายปัญหา หรือ Cuthill–McKee algorithm เป็นต้น ซึ่งหากมีการกำหนดหมายเลขโหนดที่ดีจะส่งผลให้เวลาในการคำนวณของชุดคำสั่งลดลงเป็นอย่างมาก การต่อยอดไปสู่ปัญหาในสามมิติหรือการเปลี่ยนตำแหน่งที่เกี่ยวข้องกับการหมุนและโมเมนต์ที่มีความยุ่งยากซับซ้อนกว่ามากทั้งในด้านทฤษฎีการเขียนคำสั่งไพทอน และขนาดเมทริกซ์ที่จะมีขนาดใหญ่กว่าเดิมมากส่งผลให้เวลาในการคำนวณมากตามไปด้วย ทำให้การศึกษาการเขียนและแก้ไขคำสั่งไพทอนให้มีประสิทธิภาพมากขึ้นก็เป็นสิ่งจำเป็นที่ต้องทำ และท้ายที่สุดเมื่อชุดคำสั่งดังกล่าวได้ทำการปรับปรุงแก้ไขตามตัวอย่างที่กล่าวข้างต้นไปแล้ว ก็สามารถนำชุดคำสั่งนี้ไปทำให้อยู่ในรูปแบบ GUI (Graphical user interface) เพื่อการใช้งานที่ง่ายขึ้นและเป็นมิตรกับผู้ใช้งานอื่นอีกด้วย

บรรณานุกรม

[1] สุทธิศักดิ์ พงศ์ธนาพานิช. 2560. **วิธีไฟไนต์เอลิเมนต์และการสร้างตาข่ายสามเหลี่ยม.** พิมพ์ครั้งที่ 1. กรุงเทพฯ : จุฬาลงกรณ์มหาวิทยาลัย.

[2] J.E. Akin, 2005. **Finite Element Analysis with Error Estimators: An Introduction to the FEM and Adaptive Error Analysis for Engineering Students.** 1st ed. Burlington : Elsevier Butterworth-Heinemann.

[3] David V. Hutton, 2003. **Fundamentals of Finite Element Analysis.** 1st ed. New York : McGraw-Hill.

[4] Wes McKinney, 2017. **Python for Data Analysis. Data Wrangling with Pandas, NumPy, and IPython.** 2nd ed. N.P. : O'Reilly.

[5] Robert Johansson, 2019. **Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib.** 2nd ed. New York : Apress.