МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки

«До захисту допущений»

Зав. кафедри прикладної математики

д. т. н., професор Турбал Ю. В.

«____»_____ 2023р.

КВАЛІФІКАЦІЙНА РОБОТА

Скінченноелементне покриття неоднорідних областей

Виконав: Паньов Максим Сергійович _____
(підпис)

група КН-41

Керівник: д.т.н., професор Мартинюк Петро Миколайович _____
(підпис)

Рівне – 2023

ЗМІСТ

РЕФЕРАТ

Метою кваліфікаційної роботи - створення програмного забезпечення для скінченноелементного покриття областей з тонкими включеннями.

Об’єктом досліджень є неоднорідні області.

Предметом дослідження є алгоритми скінченноелементного покриття неодорідних областей та їх (алгоритмів) програмна реалізація.

Методи вивчення – об’єктно-орієнтоване програмування , візуально-надійне програмування, мова Python.

Ключові слова: МЕТОД СКІНЧЕННИХ ЕЛЕМЕНТІВ, ЛІНІЙНІ СКІНЧЕННІ ЕЛЕМЕНТИ, КВАДРАТИЧНІ СКІНЧЕННІ ЕЛЕМЕНТИ, ПЕРЕРИВАННЯ, ОДНОВИМІРНИЙ ВИПАДОК, ДВОВИМІРНИЙ ВИПАДОК.

ВСТУП

Скінченноелементне покриття неоднорідних областей є актуальним напрямом досліджень в області чисельного моделювання і має широкий спектр застосувань у різних галузях, включаючи механіку, теплопередачу, електродинаміку та інші.

Неоднорідні області включають в себе області з різними фізичними властивостями, такими як різні матеріали, пористість, шаруватість тощо. Це створює виклики для точного моделювання цих областей, оскільки неоднорідності впливають на розподіл напружень та інших характеристик. Використання методу скінченних елементів дозволяє отримати чисельні розв'язки, які апроксимують поведінку системи у неоднорідних областях з високою точністю та ефективністю.

Однак, розробка ефективного методу скінченних елементів для неоднорідних областей є складним завданням. Вимагається розробка нових алгоритмів, апроксимаційних функцій та обробки граничних умов, що враховують неоднорідності. Також потрібно вирішити проблему обробки роздільних площин між неоднорідними областями та забезпечити точність та стабільність розрахунків.

Оскільки скінченноелементне покриття неоднорідних областей має великий потенціал для покращення точності та ефективності чисельного моделювання, результати цієї кваліфікаційної роботи можуть мати практичне застосування у різних галузях науки та техніки.

Актуальність та наукова новизна даної кваліфікаційної роботи випливає з декількох ключових факторів, а саме :

- по-перше, метод скінченних елементів є потужним і широко використовуваним інструментом для чисельного моделювання та

аналізу різноманітних фізичних процесів, в тому числі в неоднорідних областях ;

- по-друге, створення програмного забезпечення, яке автоматизує процес скінченно-елементного покриття, має велике значення для вивільнення ресурсів та розширення можливостей дослідників та інженерів при моделюванні різних систем ;

РОЗДІЛ 1

Метод скінченних елементів одновимірного випадку для областей із включеннями

1.1. Огляд робіт застосування МСЕ в різних задача

Метод скінченних елементів (МСЕ) є одним з найпоширеніших чисельних методів для моделювання фізичних процесів та розв'язання диференціальних рівнянь. Цей метод базується на розбитті складної геометрії на простіші підобласті, відомі як скінченні елементи, і побудові наближеного розв'язку на кожному елементі. Дослідження та розвиток МСЕ ведуться протягом багатьох десятиліть, і роботи на цю тему охоплюють такі аспекти:

Розвиток математичних основ: Роботи цього напрямку спрямовані на теоретичні аспекти методу скінченних елементів, такі як розробка нових апроксимаційних функцій, побудова ефективних чисельних алгоритмів, вивчення збіжності та стійкості методу, а також аналіз похибок та оцінка якості розв'язків.

Розширення на нові області застосування: Метод скінченних елементів використовується в багатьох галузях, включаючи механіку, теплопередачу, гідродинаміку, електромагнетизм, оптику, акустику та інші. Огляд робіт зосереджений на застосуванні МСЕ до конкретних фізичних процесів і вивченні їх особливостей.

Розвиток спеціалізованих методів та розширень: Існують різні варіації та розширення методу скінченних елементів, які пристосовані до специфічних задач. Наприклад, метод скінченних різниць (МСР) поєднує переваги МСЕ та методу скінченних різниць, метод скінченних об'ємів (МСО) використовує іншу формулювання рівнянь із використанням інтегральних співвідношень, метод скінченних шарів (МСШ) застосовується для розрахунків на шарованих структурах тощо.

Вдосконалення обчислювальних аспектів: Розвиток

обчислювальних технологій та доступ до потужних обчислювальних ресурсів дозволяють вдосконалювати ефективність розрахунків методом скінченних елементів. Роботи цього напрямку охоплюють розробку паралельних алгоритмів, використання високопродуктивних обчислювальних платформ, оптимізацію розподілення ресурсів і підвищення швидкодії розрахунків.

The Finite Element Method and Applications in Engineering Using ANSYS® | SpringerLink : Ця робота використовує МСЕ для прогнозування та моделювання фізичної поведінки складних інженерних систем. [1]

Explain the application of FEA in various fields. - Ques10 : В цій роботі описані застосування МСЕ в різних областях, включаючи механічну інженерію, де МСЕ використовується для стаціонарного та перехідного теплового аналізу. [2]

(PDF) APPLICATIONS OF FINITE ELEMENTS METHOD (FEM) -AN OVERVIEW: Ця робота надає загальний огляд застосувань МСЕ, потужної числової техніки для розв'язання часткових диференціальних рівнянь.

Finite Element Method with Applications in Engineering - Y. M. Desai - Google Books : Ця книга пояснює МСЕ з різними інженерними застосуваннями, щоб допомогти студентам, викладачам, інженерам та дослідникам. [3]

Ці роботи є лише декількома прикладами реалізованих робіт у сфері методу скінченних елементів. Ця галузь досліджень є широкою і продовжує розвиватися, існують багато інших цікавих робіт із різних областей застосування методу скінченних елементів.

Огляд робіт по методу скінченних елементів відображає багатогранність та актуальність цього чисельного методу. Високий рівень досліджень у цій області сприяє розширенню можливостей методу

скінченних елементів та його застосуванню для розв'язання складних наукових та технічних задач.

1.2. Огляд алгоритмів скінченноелементного покриття областей та їх програмні реалізації

1. **Delaunay Triangulation:** Це алгоритм, який максимізує мінімальний кут в трикутнику. Цей алгоритм використовується для створення трикутникових сіток. [4]

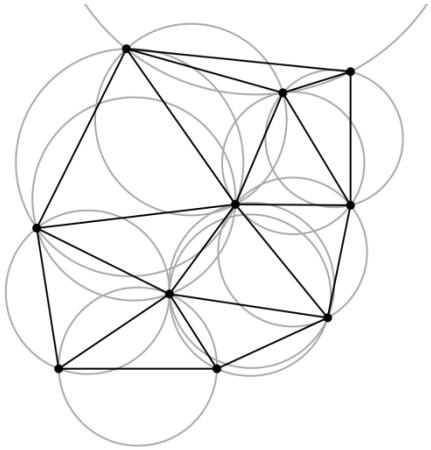

Рис.1.1. Тріангуляція Делоне

1. **Advancing Front Method:** Цей алгоритм використовується для генерації сіток в 2D і 3D. Це здійснюється шляхом поступового перенесення "фронту" через область, що аналізується, з одночасним додаванням нових елементів до сітки. [5]

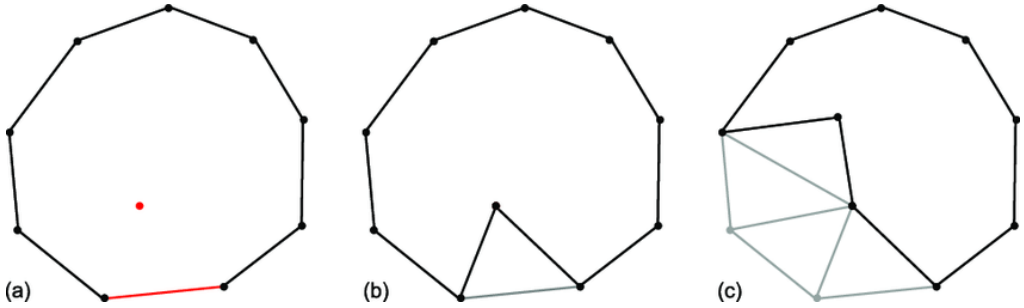

Рис.1.2. Метод «Advancing Front»

2. **Quadtree (2D) and Octree (3D) Refinement:** У цих алгоритмах сітка перебудовується в областях, де потрібно більше точності, поділяючи клітини на менші або навпаки. [6]

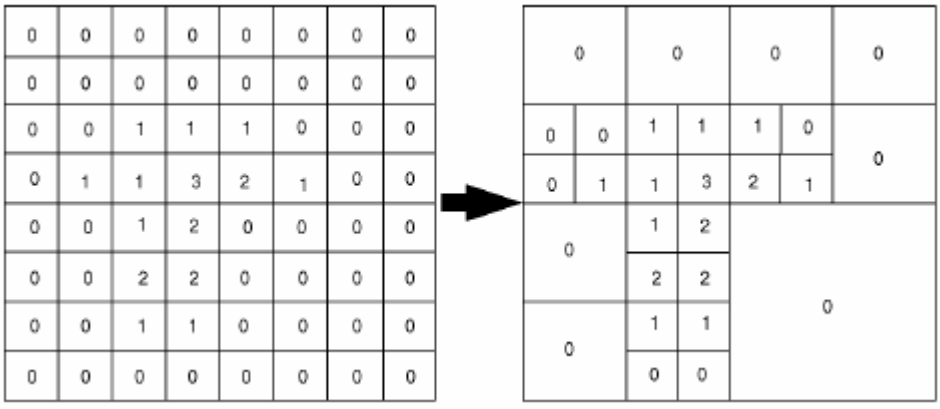

Рис.1.3. Quadtree (2D) and Octree (3D) Refinement

Програмні реалізації МСЕ:

- **ANSYS:** Потужне ПЗ, яке включає можливості для проведення структурного аналізу, аналізу теплопередачі, аналізу потоків рідини і багато іншого. Можна придбати платні версії, проте у доступі також є безкоштовні пробні версії. [7]

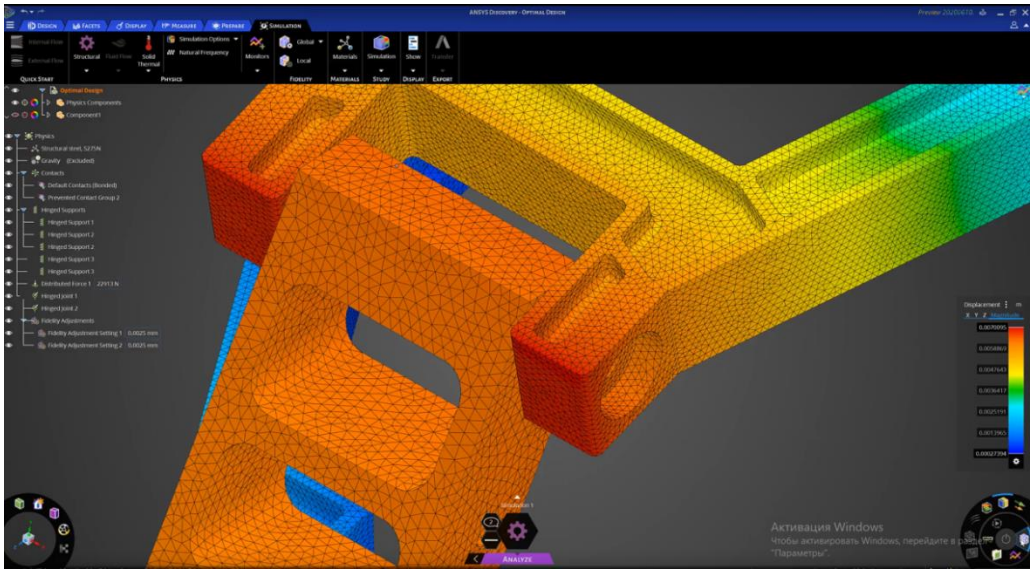

Рис. 1.4. Інтерфейс ANSYS

- **Abaqus:** ПЗ, яке використовується для проведення розширеного скінченноелементного аналізу. Його сильні сторони включають можливість моделювання нелінійних матеріалів і складних контактів. ПЗ платне. [8]

- **COMSOL Multiphysics:** ПЗ, яке використовується для моделювання інженерних проблем, які включають кілька фізичних полів одночасно. Доступна лише платна версія. [9]

1.3. Важливість задач в неоднорідних областях

МСЕ став незамінним інструментом для чисельного розв'язання широкого спектра фізичних задач. Проте, щоб ефективно використовувати МСЕ , область, яка має бути аналізована, спочатку повинна бути поділена на менші елементи, що формують скінченноелементне покриття або "сітку". Процес розподілу або "дискретизації" області є критичним кроком в МСЕ , і якість цього покриття може значно вплинути на точність і ефективність чисельного розв'язку.

Актуальність розвитку алгоритмів для скінченноелементного покриття в неоднорідних областях можна підкреслити кількома ключовими пунктами:

1. Потужний інструмент для моделювання: Метод скінченних елементів став одним з найпотужніших інструментів для чисельного моделювання та аналізу різних фізичних процесів. Це особливо важливо при роботі з неоднорідними областями, де властивості матеріалу, геометрія або інші фактори можуть варіюватися в просторі. Здатність точно моделювати такі системи з неоднорідними властивостями є критично важливою для численних застосувань, включаючи інженерію, фізику, математику, медицину та ін.

2. Автоматизація та ефективність: Розробка програмного забезпечення, яке автоматизує процес скінченноелементного покриття, значно вивільняє ресурси, що раніше потребували великої кількості ручного втручання та інтелектуального навантаження. Це не тільки підвищує ефективність процесу, але і розширює можливості дослідників та інженерів, дозволяючи їм сконцентруватися на більш складних і незвичайних аспектах моделювання.

3. Вища точність моделювання: Створення точного скінченноелементного покриття є критично важливим для отримання точних чисельних результатів. В неоднорідних областях, де властивості можуть

значно варіюватися, ефективні алгоритми для створення таких покриттів можуть значно підвищити якість моделювання.

Загалом, існує постійна потреба в розвитку та удосконаленні алгоритмів для створення скінченноелементних покриттів в неоднорідних областях, оскільки вони можуть принести значний внесок в якість та ефективність чисельного моделювання різноманітних фізичних систем.

РОЗДІЛ 2

ЕЛЕМЕНТИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОД СКІЧЕННИХ ЕЛЕМЕНТІВ ОДНОВИМІРНОЇ ОБЛАСТІ З ВКЛЮЧЕННЯМИ

2.1. Алгоритм роботи МСЕ одновимірного випадку з включеннями

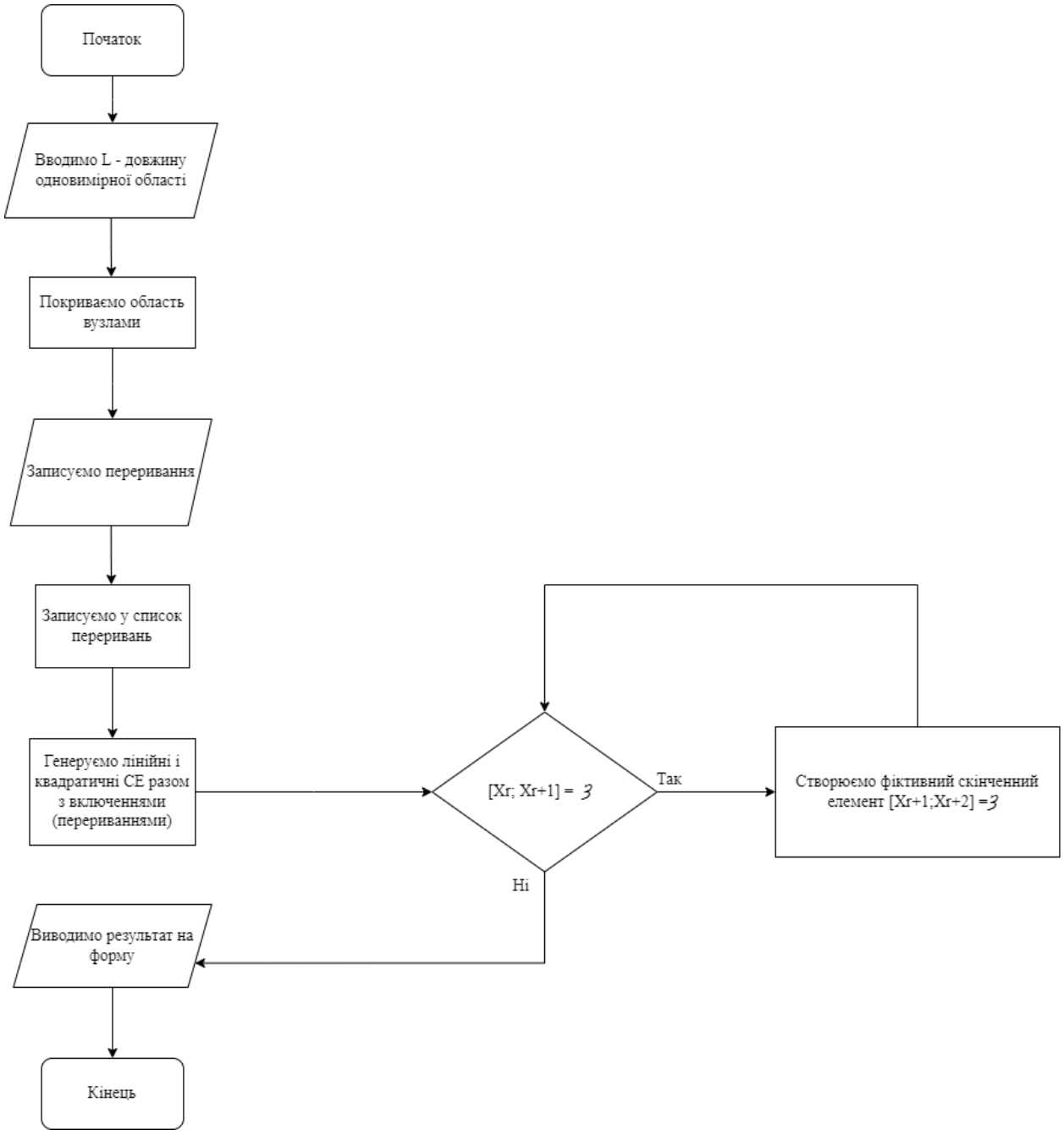

Рис.2.1 Алгоритм роботи МСЕ одновимірного випадку з включеннями

2.2. Інтерфейс користувача

Програмна реалізація МСЕ здійснена в середовищі програмування Python (версія 3.11.1). Вигляд головного меню програми показано на рис.2.2.

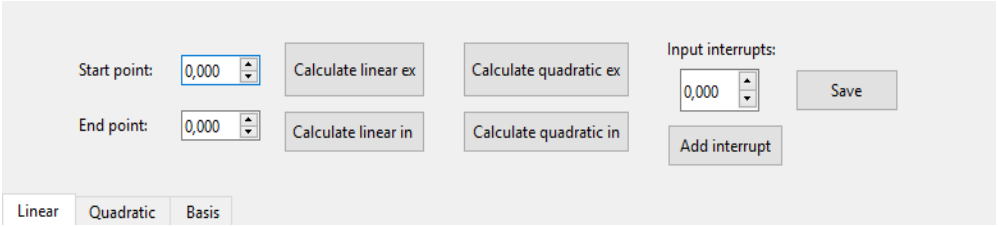

Рис. 2.2. Головне меню програми

Спочатку потрібно вказати область (L), яку ми покриємо скінченними елементами (пам'ятаємо, що h – довжина СЕ за замовчуванням рівна 0,005) . Для цього вводимо початок і кінець нашої області відповідно в пунктах меню «Start point» і «End point» (рис.2.2.) і на даній формі по кліку на кнопку «Calculate linear ex» генеруємо вузлову сітку для лінійних СЕ (рис.2.3.).

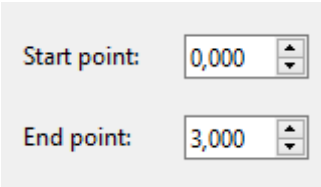

Рис. 2.3. Довжина області

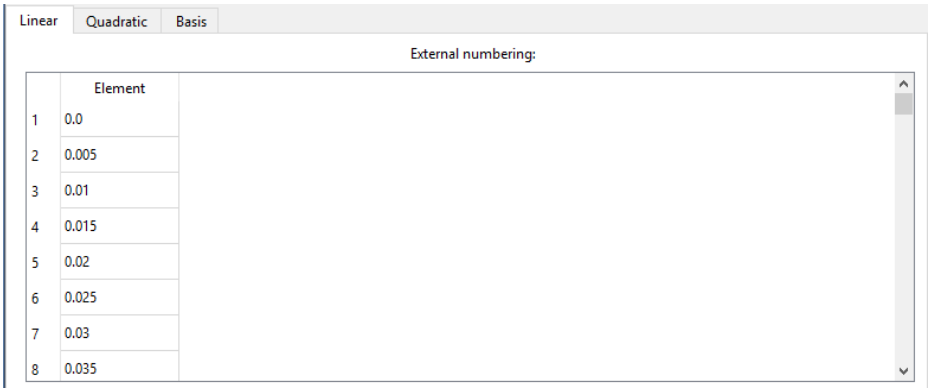

Рис. 2.4. Генерація вузлової сітки лінійних СЕ

Таку ж вузлову сітку ми можемо згенерувати для квадратичних СЕ, натиснувши кнопку «Calculate quadratic ex» на панелі головного меню та перейшовши на сторінку «Quadratic» (рис.2.2.) . Відмінність між лінійними і квадратичними СЕ лиш наявністю внутрішнього вузла у останніх (рис. 2.5).

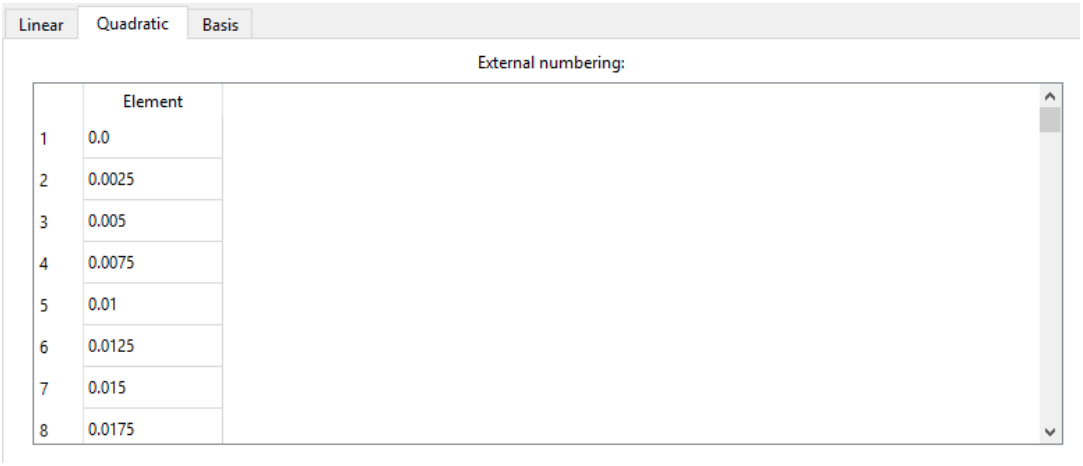

Рис. 2.5. Генерація вузлової сітки квадратичних СЕ

Далі, щоб обчислити лінійні СЕ, нам необхідно задати переривання у відділі, який позначений «Input interrupts». Пишемо число всередині віджета і натискаємо кнопку головного меню «Add interrupt» - для збереження переривання у спеціально відведеному списку (рис.2.6.) для усіх переривань. Програмою передбачено, що переривання не можуть попадати на початок і кінець відрізка. Коли усі переривання будуть задані, ми можемо зберегти їх і вивести на форму(рис.2.7.) , натиснувши кнопку «Save». Зауважте, що після цього ми не зможемо добавляти ніяких переривань, для коректної роботи нашої програми.

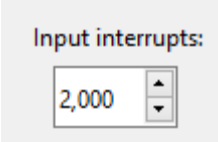

Рис. 2.6. Запис переривання

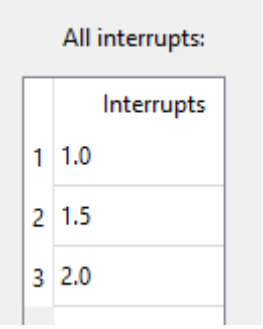

Рис. 2.7. Список усіх переривань

При невиконанні цієї умови – функції для генерації лінійних і квадратичних СЕ будуть недоступними для виконання.

Тепер на даному етапі нашої роботи ми можемо згенерувати лінійні СЕ з нашої вузлової сітки і уникнувши переривань в нумерації, натиснувши кнопку «Calculate linear in» (рис. 2.2.) . Результати будуть виведені на форму (рис. 2.8.).

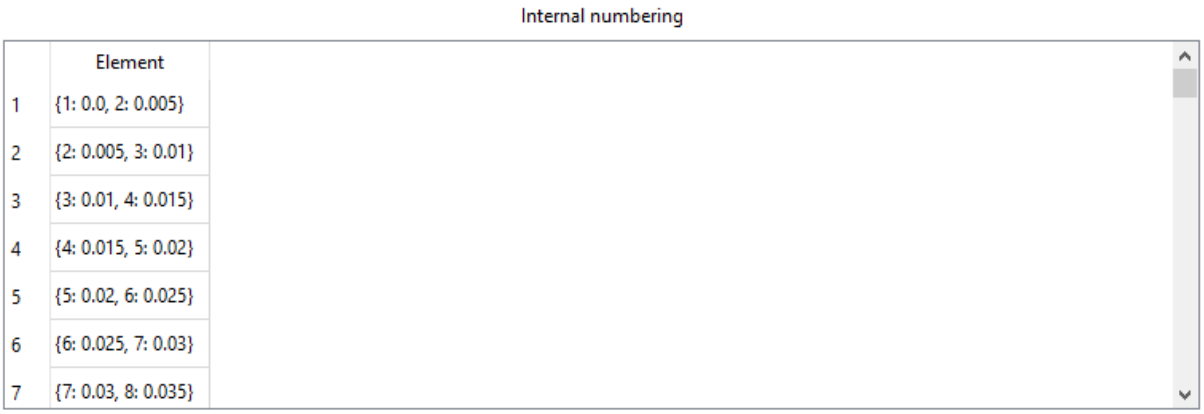

Рис. 2.8. Генерація лінійних СЕ

Таку ж генерацію ми можемо виконати для квадратичних СЕ, натиснувши кнопку «Calculate quadratic in» (рис. 2.9.).

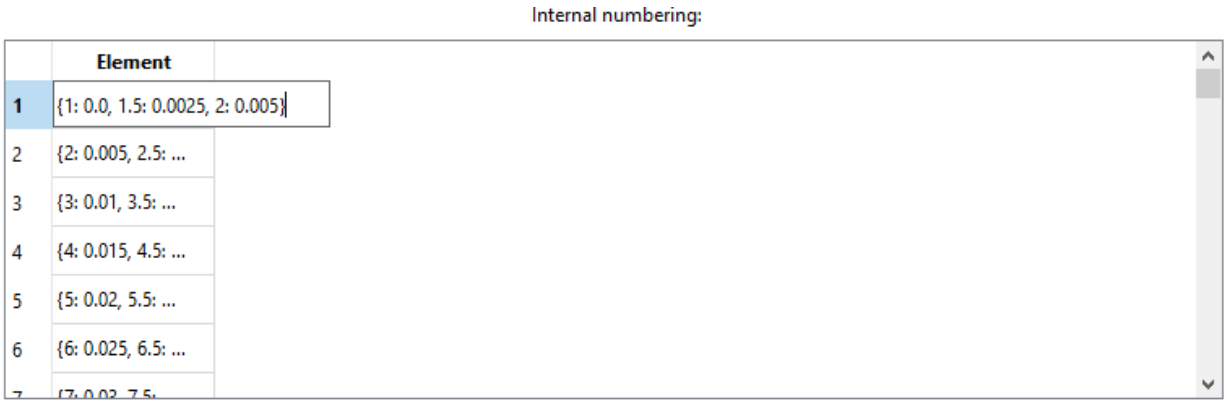

Рис. 2.9. Генерація квадратичних СЕ

2.3. Програмна реалізація основних станів алгоритму

Програма являє собою 4 основних етапи: покриття області вузлами, внесення переривань, генерація лінійних СЕ, генерація квадратичних СЕ.

Для реалізації першого етапу написана функція покриття області вузлами. Також визначається кількість вузлів.

Для лінійних СЕ:

```
def external_numbering_linear_elements(self,num1,num2):
                    i = 0
                    element = 0
                    keys = []  # ключі по яких шукаємо
елементи
                    elements = []  # елементи
```

```
                        idata.yp = num1
                        idata.yk = num2
                        if (idata.h > num1):  # функція для
зовнішної нумерації одновимірного випадку
                            if (idata.h  < num2):
                                while num1 < num2:
                                    element = num1

elements.append(round(float(element),3))
                                    num1 += idata.h
                                    i += 1
                                    keys.append(i)
                        # зовнішня нумерація одновимірного випадку
                        idata.generation = dict(zip(keys,
elements))
                        print("\n")
                        print(idata.generation)

self.tableWidget_for_linear_external_numbering.setRowCount((len(
keys)))

self.tableWidget_for_linear_external_numbering.setColumnCount(1)

self.tableWidget_for_linear_external_numbering.setHorizontalHead
erLabels(('Element','Elements'))
                        row = 0
                        for el in idata.generation:

self.tableWidget_for_linear_external_numbering.setItem( row, 0 ,
QtWidgets.QTableWidgetItem(str(idata.generation[el])))
                            row = row + 1
```

Для квадратичних СЕ:

```
def external_numbering_quadratic_elements(self,numer1,numer2):
        i = 0
        element = 0
        element1= 0
        keys1 = []  # ключі по яких шукаємо елемент
        elements1 = []  # елементи
        if (idata.h  > numer1):  # функція для зовнішної
нумерації квадратичного випадку
            if (idata.h  < numer2):
                while numer1 < numer2:
                    element = numer1
                    element1 = numer1 + idata.h
                    elements1.append(round(float(element),3))
                    i += 1
                    keys1.append(i)
                    elements1.append(round(float((element +
element1)/2),4))
                    i += 1
                    keys1.append(i)
                    numer1 += idata.h
        # зовнішня нумерація квадратичного випадку
        idata.generation1 = dict(zip(keys1, elements1))
        idata.generation1.pop(len(idata.generation1))
        print("\n")
        print(idata.generation1)

self.tableWidget_for_quadratic_external_numbering.setRowCount(le
n(idata.generation1))
self.tableWidget_for_quadratic_external_numbering.setColumnCount
(1)
self.tableWidget_for_quadratic_external_numbering.setHorizontalH
eaderLabels(('Element','Elements'))
```

```
        row = 0
       for el in idata.generation1:
self.tableWidget_for_quadratic_external_numbering.setItem( row,
0 , QtWidgets.QTableWidgetItem(str(idata.generation1[el])))
            row = row + 1
```

Для другого етапу, а саме для внесення переривань, написано дві функції, що відповідають за додавання та зберігання переривань у списку відповідно .

```
def add_interrupts(self,count):
        if(count>=idata.yp and count<idata.yk):
            idata.check.append(count)
            for cheking in range(len(idata.check)):
                 if (idata.check.count(cheking) > 1 ):
                    idata.check.remove(cheking)
        else:
           error = QMessageBox()
           error.setWindowTitle("Error!")
           error.setText("Input another interrupt")
           error.exec()
```

```
def save_interrupts(self):
      if(len(idata.check)==0):
           error1 = QMessageBox()
           error1.setWindowTitle("Error!")
```

```
                error1.setText("Input iterrupts")
                error1.exec()
            else:
                # масиви переривань для лінійних скінченних
елементів
                for i in idata.check :
                    idata.yk12 = list(set(idata.check))
                idata.yk12.sort()
                print("\n")
                print("yk12 = ", idata.yk12)
                for i in range(len(idata.yk12)):
                    idata.yk13.append(idata.yk12[i])
                    if (i == 0):
                        idata.yk13.append(idata.yk12[i])
                idata.yk13.sort()
                print("\n")
                print("yk13 = ", idata.yk13)
                # масиви переривань для квадратичних
скінченних елементів і базису
                for i in idata.yk12 :
                    idata.interrupt12 = list(set(idata.yk12))
                    idata.inter_basis = list(set(idata.yk12))
                idata.interrupt12.sort()
                idata.inter_basis.sort()
                print("\n")
                print("interrupt12=",idata.interrupt12)
                for i in range(len(idata.interrupt12)):

idata.interrupt13.append(idata.interrupt12[i])
                    if (i == 0):

idata.interrupt13.append(idata.interrupt12[i])
```

```
                    idata.interrupt13.sort()
                    print("\n")
                    print("interrupt13=",idata.interrupt13)
                    idata.inter_basis = tuple(idata.inter_basis)
                    row = 0

self.tableWidget_show_all_interrupts.setRowCount(len(idata.yk12)
)
self.tableWidget_show_all_interrupts.setColumnCount(1)
self.tableWidget_show_all_interrupts.setHorizontalHeaderLabels((
'Interrupts','Elements'))
            for interrupt in idata.yk12:

self.tableWidget_show_all_interrupts.setItem(row,0
,QtWidgets.QTableWidgetItem(str(interrupt)))
self.tableWidget_show_all_interrupts.setItem(row,
1,QtWidgets.QTableWidgetItem(str(interrupt)))
                        row = row + 1
```

Наступний етап генерації лінійних і квадратичних скінченних елементів має подібні функції:

Лінійні СЕ:

```
def internal_numbering_linear_elements(self):
        if(len(idata.yk12)!= 0 and len(idata.yk13)!=0):
            stop = len(idata.generation)
            item = 0
            if(len(idata.generation) != 0):  # функція  для
внутрішньої нумерації одновимірного випадку
                for el in range(1, len(idata.generation)):
                    numer = {}  # словник в який поміщаємо два
елемента
                    if (idata.generation[el] < idata.yk13[0]):
```

```
                        numer = {el: idata.generation[el], el+1:
idata.generation[el+1]}
                        idata.numer1[el] = numer
                    elif (idata.generation[el] ==
idata.yk12[0]):
                        item = el
                        numer = {el: idata.generation[el], el+1:
idata.generation[el]}
                        idata.numer1[el] = numer
                        if (len(idata.yk12) > 1):
                            idata.yk12.pop(0)
                    elif (idata.generation[el] > idata.yk13[0]):
                        item = el
                        numer = {el: idata.generation[el-1],
el+1: idata.generation[el]}
                        idata.numer1[el] = numer
                        if (len(idata.yk13) > 1):
                            idata.yk13.pop(0)
                    elif (el == stop):
                        numer = {el+1: idata.generation[el]}
                        idata.numer1[el] = numer
            print("\n")
            print(idata.numer1)
            idata.yk12 = tuple(idata.yk12)
            idata.yk13 = tuple(idata.yk13)
            print("\n")
            print(idata.yk12)
            print("\n")
            print(idata.yk13)

self.tableWidget_for_linear_internal_numbering.setRowCount((len(
idata.numer1)))
self.tableWidget_for_linear_internal_numbering.setColumnCount(1)
```

```
self.tableWidget_for_linear_internal_numbering.setHorizontalHead
erLabels(('Element','Elements'))

            row = 0

            for elements in idata.numer1:
self.tableWidget_for_linear_internal_numbering.setItem( row, 0 ,
QtWidgets.QTableWidgetItem(str(idata.numer1[elements])))

                row = row + 1
```

Квадратичні СЕ:

```
def internal_numbering_quadratic_elements(self):

             if(len(idata.yk12)!= 0 and len(idata.yk13)!=0):

                item = 0

                stop = len(idata.generation)

                if(len(idata.generation) != 0):  # функція  для
внутрішньої нумерації квадратичного випадку

                    for el in range(1, len(idata.generation)):

                        numer = {}  # словник в який поміщаємо
три елемента

                        if (idata.generation[el] <
idata.interrupt13[0]):

                            numer = {el: idata.generation[el],
el+0.5: round((idata.generation[el] +
idata.generation[el+1])/2,4), el+1: idata.generation[el+1]}

                            idata.numer2[el] = numer

                        elif (idata.generation[el] ==
idata.interrupt12[0]):

                            item = el

                            numer = {el: idata.generation[el],
el+0.5: round((idata.generation[el] +
idata.generation[el])/2,4), el+1: idata.generation[el]}

                            idata.numer2[el] = numer

                            if (len(idata.interrupt12) > 1):

                                idata.interrupt12.pop(0)
```

```
                        elif (idata.generation[el] >
idata.interrupt13[0]):

                            item = el

                            numer = {el: idata.generation[el-1],
el+0.5: round((idata.generation[el-1] +
idata.generation[el])/2,4), el+1: idata.generation[el]}

                            idata.numer2[el] = numer

                            if (len(idata.interrupt13) > 1):

                                idata.interrupt13.pop(0)

                        elif (el == stop):

                            numer = {el+1: idata.generation[el]}

                            idata.numer2[el] = numer
self.tableWidget_for_quadratic_inernal_numbering.setRowCount((le
n(idata.numer2)))
self.tableWidget_for_quadratic_inernal_numbering.setColumnCount(
1)
self.tableWidget_for_quadratic_inernal_numbering.setHorizontalHe
aderLabels(('Element','Elements'))

                    row = 0

                    for elements in idata.numer2:
self.tableWidget_for_quadratic_inernal_numbering.setItem( row, 0
, QtWidgets.QTableWidgetItem(str(idata.numer2[elements])))

                        row = row + 1
```

РОЗДІЛ 3

ЕЛЕМЕНТИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОД РУХОМОГО ФРОНТУ ТРІАНГУЛЯЦІЇ ДВОВИМІРНОЇ ОБЛАСТІ З ВКЛЮЧЕННЯМИ

3.1 Алгоритм роботи програми методу рухомого фронту з включеннями

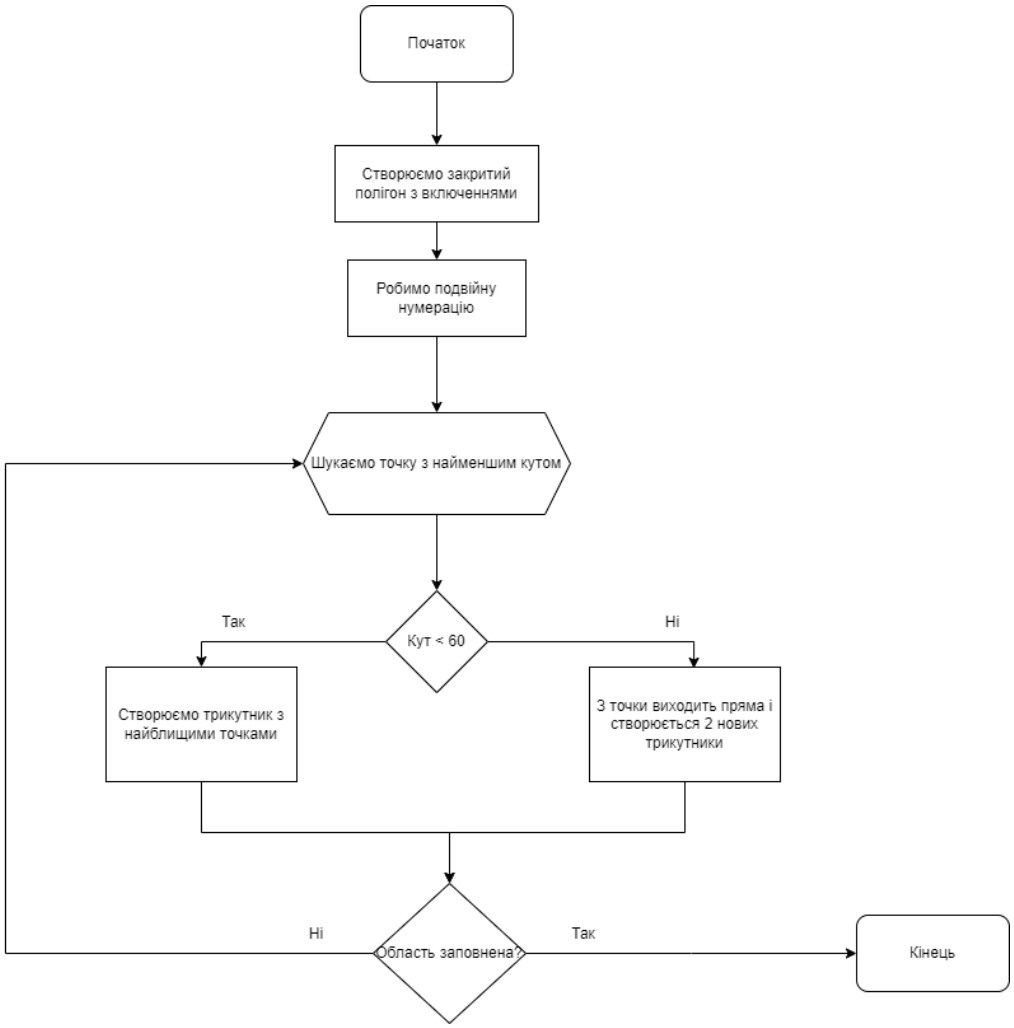

Рис.3.1 Алгоритм роботи програми для методу рухомого фронту

3.2. Інтерфейс користувача

Програмна реалізація методу рухомого фронту здійснена в середовищі програмування Python (версія 3.9.0) . Головного меню програми показано на рис.3.2.

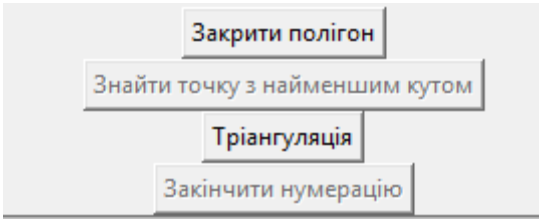

Рис. 3.2. Головне меню

Для роботи нам потрібно створити закриту область, яку ми будемо тріангулювати. Натискаючи правою кнопкою миші по формі, ми будемо ставити точки, які будуть між собою з’єднуватись лініями і нумеруватись. Цей процес має такий вигляд як на рис. 3.3.

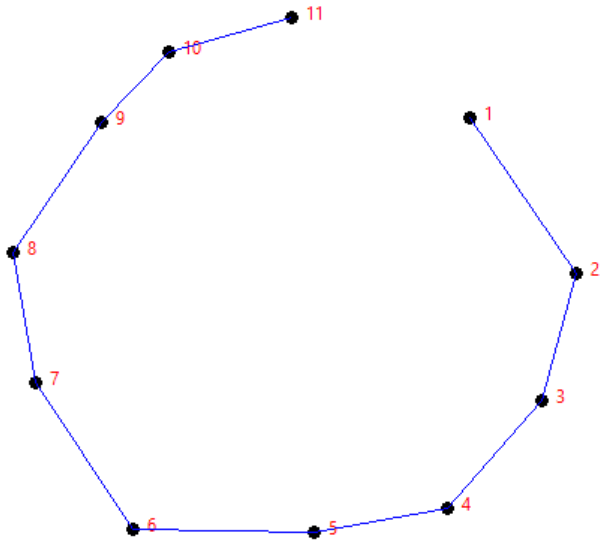

Рис. 3.3. Створення закритого полігону

Закрити область (сполучити 11 точку з 1) ми можемо натиснувши кнопку «Закрити полігон». Далі нам потрібно поставити ще декілька точок в середині полігона як на рис. 3.4.

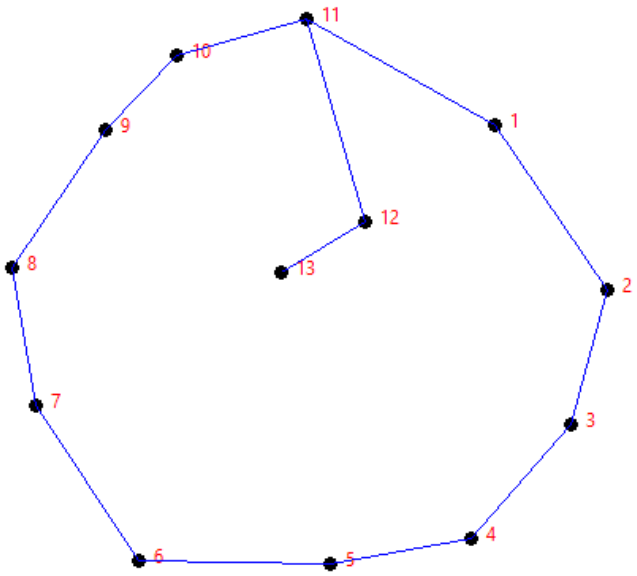

Рис. 3.4. Закритий полігон

На даному етапі роботи ми можемо зробити подвійну нумерацію, натиснувши кнопку головного меню «Закінчити нумерацію», для точок, які всередині полігона і закінчуючи точкою, яка закрила полігон. Це потрібно для коректної роботи програми.

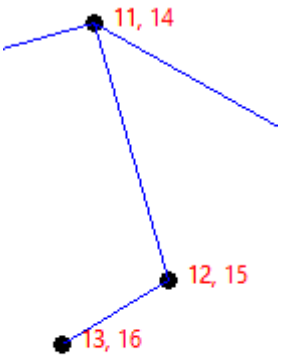

Рис.3.5. Подвійна нумерація

Тепер, коли наш закритий полігон готовий, ми можемо починати основні дії. Тобто тепер за алгоритмом ми шукаємо найменший кут, для цього натискаємо кнопку «Знайти точку з найменшим кутом», а потім покриваємо область трикутниками, натиснувши кнопку «Тріангуляція». Основний задум в тому, що коли ми знаходимо точку, яка має найменший

кут < 60 градусів, то просто створюємо трикутник, але коли знаходимо точку і її кут > 60, то з неї виходить пряма, яка ділить цей кут пополам, і довжина цього відрізка дорівнює півсумі довжин відрізків з яких складається сам кут. Вигляд це буде мати як на рис. 3.6.

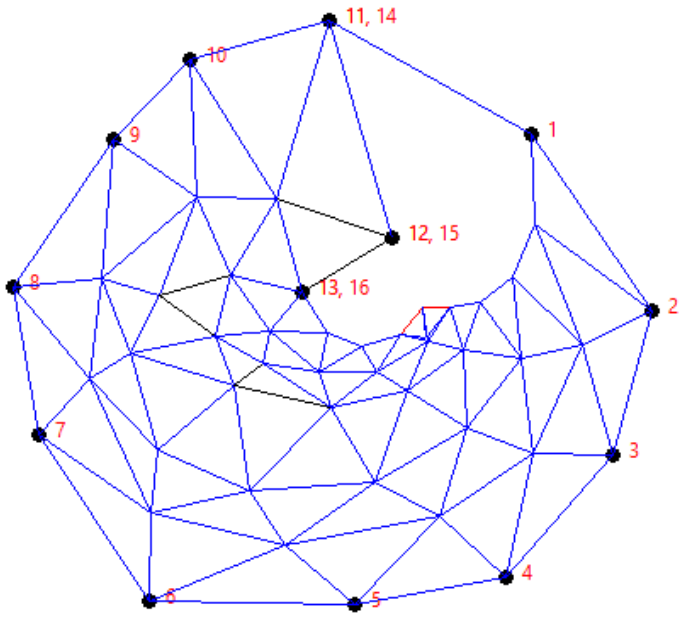

Рис. 3.6. Тріангуляція полігону

3.3. Програмна реалізація основних станів алгоритму

Програма являє собою 3 основних етапи: створення закритого полігону, знаходження точки з найменшим кутом, тріангуляція закритої області.

Для реалізації першого етапу реалізовано 2 функції , а саме:

- **add_point(self, event):** Ця функція викликається при клацанні мишею на полі для малювання. Вона додає точку до списку точок та малює нову точку та лінію, яка з'єднує її з попередньою точкою.

• **close_polygon(self):** Ця функція замикання полігону. Вона з'єднує останню та першу точки, а потім змінює стан кнопок.

```
    def add_point(self, event):
            if self.polygon_closed:
                polygon_path = Path(self.points)
                if not polygon_path.contains_point((event.x, event.y)):
                    return

            x, y = event.x, event.y
            self.points.append((x, y))
            self.canvas.create_oval(x - self.point_radius, y - self.point_radius,
                                    x + self.point_radius, y + self.point_radius,
                                    fill='black')
            self.canvas.create_text(x + 10, y - 10, text=str(len(self.points)), fill='red', anchor='nw')
            if len(self.points) > 1:
                line = self.canvas.create_line(self.points[-2], self.points[-1], fill='blue')
                self.lines.append(line)

            self.canvas.update()

            if self.polygon_closed:

self.finish_numbering_button.config(state='normal')
        def close_polygon(self):
            if len(self.points) > 2 and not self.polygon_closed:
```

```
                    line = self.canvas.create_line(self.points[-
1], self.points[0], fill='blue')
                    self.lines.append(line)
                    self.polygon_closed = True
                    self.starting_point_index = len(self.points) -
1
                    self.close_button.config(state='disabled')
self.smallest_angle_button.config(state='normal')
                    self.triangulate_button.config(state='normal')
                    self.close_button.config(state='disabled')
self.finish_numbering_button.config(state='normal')
                    self.triangulate_button.config(state='normal')
```

Етап для знаходження точки з найменшим кутом реалізовано у двох функціях:

- **angle(self, p1, p2, p3):** Ця функція обчислює кут між трьома точками (p2 - центральна точка).

- **find_point_with_smallest_angle(self):** Ця функція шукає точку з найменшим кутом в полігоні. Вона також малює лінії, які формують цей кут.

```
def angle(self, p1, p2, p3):
        # Vector from p1 to p2
        v1 = [p2[0]-p1[0], p2[1]-p1[1]]
        # Vector from p2 to p3
        v2 = [p3[0]-p2[0], p3[1]-p2[1]]
        # Calculate the dot product of v1 and v2
```

```
                dot_product = v1[0]*v2[0] + v1[1]*v2[1]
                # Calculate the magnitude of v1 and v2
                mag_v1 = np.sqrt(v1[0]**2 + v1[1]**2)
                mag_v2 = np.sqrt(v2[0]**2 + v2[1]**2)
                # Calculate the angle between v1 and v2
                cos_angle = dot_product / (mag_v1 * mag_v2)
                angle = np.arccos(cos_angle)
                return angle
```

```
    def find_point_with_smallest_angle(self):
                if len(self.points) < 3:
                    return None, None, None

                def angle(a, b, c):
                    ba = np.array(a) - np.array(b)
                    bc = np.array(c) - np.array(b)

                    cos_angle = np.dot(ba, bc) /
(np.linalg.norm(ba) * np.linalg.norm(bc))
                    angle = np.arccos(cos_angle)

                    return angle

                smallest_angle = np.inf
                smallest_angle_point = None
                smallest_angle_index = None

                for i in range(1, len(self.points) - 1):
                    current_angle = angle(self.points[i-1],
self.points[i], self.points[i+1])
```

```
                    if current_angle < smallest_angle:
                        smallest_angle = current_angle
                        smallest_angle_point = self.points[i]
                        smallest_angle_index = i

                # Очистка попередніх ліній кута
                for line in self.angle_lines:
                    self.canvas.delete(line)
                self.angle_lines = []

                # Якщо знайдено точку з найменшим кутом, створити лінії до сусідніх точок
                if smallest_angle_index is not None:
                    prev_point_index = smallest_angle_index - 1 if smallest_angle_index > 0 else len(self.points) - 1
                    next_point_index = smallest_angle_index + 1 if smallest_angle_index < len(self.points) - 1 else 0

                    prev_point = self.points[prev_point_index]
                    next_point = self.points[next_point_index]

                    line1 = self.canvas.create_line(prev_point, smallest_angle_point, fill='red')
                    line2 = self.canvas.create_line(smallest_angle_point, next_point, fill='red')

                    self.angle_lines.append(line1)
                    self.angle_lines.append(line2)

                # Додати вивід значення найменшого кута в градусах
                print("Найменший кут:", np.degrees(smallest_angle))
```

```
            return smallest_angle_point, smallest_angle_index,
smallest_angle
```

Найвищий етап тріангуляції закритої області зроблений у декількох пов’язаних між собою функціях:

- **on_triangulation_button_click(self):** Ця функція викликається, коли натискається кнопка "Тріангуляція". Вона обчислює трикутники на основі кутів та додає нові точки, якщо необхідно.
- **add_triangle(self, p1, p2, p3):** Ця функція створює новий трикутник за трьома точками. Вона також малює цей трикутник на полі для малювання та перевіряє, чи перетинається новий трикутник з будь-яким з існуючих трикутників.

```
def on_triangulation_button_click(self):
        self.original_points = list(self.points)
        smallest_angle_point, smallest_angle_index,
smallest_angle = self.find_point_with_smallest_angle()
        if smallest_angle_point is not None:
            p1 = self.points[smallest_angle_index - 1]
            p2 = self.points[smallest_angle_index]
            p3 = self.points[smallest_angle_index + 1]
            if np.degrees(smallest_angle) < 60:  # Check if the
angle is less than 60 degrees
                self.add_triangle(p1, p2, p3)

                # Remove the point from the polygon
                del self.points[smallest_angle_index]
```

```
                # Add a new line
                new_line =
self.canvas.create_line(self.points[smallest_angle_index - 1],
self.points[smallest_angle_index % len(self.points)],
fill='blue')
                self.lines.insert(smallest_angle_index - 1,
new_line)
            else:
                # Compute the length of the new line
                midline_length = (np.linalg.norm(np.array(p1) -
np.array(p2)) + np.linalg.norm(np.array(p3) - np.array(p2))) / 2

                # Compute the coordinates of the new point
                mid_point = ((p1[0] + p3[0]) / 2, (p1[1] +
p3[1]) / 2)

                # Direction from smallest_angle_point to
mid_point
                direction = np.array(mid_point) - np.array(p2)
                direction = direction /
np.linalg.norm(direction)

                # New point
                new_point = tuple(np.array(p2) + midline_length
* direction)

                # Add the new point to the list of points
```

```
                self.points.insert(smallest_angle_index + 1,
new_point)


                # Add new lines
                self.canvas.create_line(p2, new_point,
fill='blue')
                self.canvas.create_line(p1, new_point,
fill='blue')
                self.canvas.create_line(p3, new_point,
fill='blue')


                # Do not delete the original point with the
smallest angle from the canvas
                del self.points[smallest_angle_index]
                # Do not delete the line from the canvas
                del self.lines[smallest_angle_index - 1]


    def add_triangle(self, p1, p2, p3):
        if self.is_valid_triangle(p1, p2, p3):
            triangle = self.canvas.create_polygon(p1, p2, p3,
fill='', outline='black')
            self.triangle_lines.append(triangle)
```

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Madenci, E., & Guven, I. (2006). The Finite Element Method and Applications in Engineering Using ANSYS. Springer.
2. Muñoz, G. A. (2010). The Finite Element Method and Applications in Engineering Using ANSYS. Springer.
3. Desai, Y. M. (2013). Finite Element Method with Applications in Engineering. Pearson.
4. Wikipedia. (2023). Delaunay triangulation. https://en.wikipedia.org/wiki/Delaunay_triangulation
5. Wikipedia. (2023). Octree. https://en.wikipedia.org/wiki/Octree
6. ANSYS Fluent. (2023). https://www.ansys.com/products/fluids/ansys-fluent
7. Abaqus. (2023). https://www.3ds.com/products-services/simulia/products/abaqus/
8. COMSOL Multiphysics. (2023). https://www.comsol.com/
9. https://chat.openai.com/c/e85f6882-1149-4bf1-baa3-b45371fd7990