Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Санкт-Петербургский государственный архитектурно-строительный университет»**

Факультет строительный

Кафедра информационных систем и технологий

**Курсовая работа**

**Панель двоякой кривизны**

Обучающийся Мельниченко Д.С.

Направление подготовки: 01.03.02 – Прикладная математика и информатика

Группа: ПМИб-4

Оценка

Дата

Преподаватель

Семенов А.А., к.т.н., доцент, доцент кафедры ИСТ

*(подпись) (Ф.И.О., должность)*

Санкт-Петербург

2023 г.

**Лист задания**

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АРХИТЕКТУРНО- СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет инженерной экологии и городского хозяйства

Направление 01.03.02 Прикладная математика и информатика

Направленность (профиль) 01.03.02 Прикладная математика и информатика

Кафедра информационных систем и технологий

ЗАДАНИЕ

НА ВЫПОЛНЕНИЕ КУРСОВОЙ РАБОТЫ

Студенту Мельниченко Д.С.

Группа ПМИб-4

Тема КР «Анализ прочности и устойчивости тонкостенной оболочной конструкции типа двоякой кривизны»

Срок сдачи законченной работы 28.12.2023

Исходные данные к работе:

Дана оболочка двоякой кривизны с *толщиной* h = 0.22 [мм], двумя *линейными размерами* вдоль x-координаты ,

*Нагрузка* вдоль х-координаты – изменяющаяся по параболическому закону с возрастанием к центру, вдоль y-координаты –аналогичная,

*Материалы* – углепластик Т300/976,Сталь С345,Орг.стекло.

*Критерии* для проверки прочности – Мизеса и Хоффмана.

Содержание расчетно-пояснительной записки: цель – анализ прочности и устойчивости тонкостенной оболочной конструкции в геометрически нелинейной постановке модели Тимошенко-Рейсснера. Задачи: ознакомление с моделью Тимошенко-Рейснера, разработка расчетного алгоритма, написание программы, проведение серии вычислительных экспериментов и анализ полученных результатов

Перечень графического материала рисунки, таблицы, программный код

Дата выдачи задания 10.10.23

Студент Мельниченко Д.С.

подпись

Руководитель КР: Семенов А.А.. к.т.н., доцент, доцент кафедры ИСТ

подпись

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 3](#_Toc153924432)

[ВВЕДЕНИЕ 4](#_Toc153924433)

[ПОСТАНОВКА ЗАДАЧИ 5](#_Toc153924434)

[МАТЕМАТИЧЕСКАЯ МОДЕЛЬ 6](#_Toc153924435)

[ГЕОМЕТРИЧЕСКИЕ СООТНОШЕНИЯ 6](#_Toc153924436)

[ФИЗИЧЕСКИЕ СООТНОШЕНИЯ 7](#_Toc153924437)

[УСИЛИЯ И МОМЕНТЫ 7](#_Toc153924438)

[ФУНКЦИОНАЛ ПОЛНОЙ ЭНЕРГИИ ДЕФОРМАЦИИ 8](#_Toc153924439)

[АЛГОРИТМЫ И МЕТОДЫ 10](#_Toc153924440)

[АНАЛИЗ ПРОЧНОСТИ ОБОЛОЧКИ 14](#_Toc153924441)

[КРИТЕРИЙ ПРЕДЕЛЬНОГО СОСТОЯНИЯ ДЛЯ ИЗОТРОВНОГО МАТЕРИАЛА 14](#_Toc153924442)

[КРИТЕРИЙ МИЗЕСА 15](#_Toc153924443)

[КРИТЕРИЙ ПРЕДЕЛЬНОГО СОСТОЯНИЯ ДЛЯ ОРТОТРОПНОГО МАТЕРИАЛА 16](#_Toc153924444)

[ВХОДНЫЕ ДАННЫЕ 17](#_Toc153924445)

[РАСЧЕТЫ 20](#_Toc153924446)

[ЗАКЛЮЧЕНИЕ 22](#_Toc153924447)

[СПИСОК ЛИТЕРАТУРЫ 23](#_Toc153924448)

[ПРИЛОЖЕНИЕ 24](#_Toc153924449)

# ВВЕДЕНИЕ

В строительстве тонкостенные оболочечные конструкции используются для покрытия помещений больших площадей, таких как стадионы, ангары, цирки, аэропорты.

В архитектурной практике используются оболочки конические, сферические, цилиндрические, пологие, торообразные, и другие. Как правило, конструкции данного типа подвергаются воздействию различных нагрузок, поэтому возникает необходимость исследования их устойчивости.

В данной работе производится исследование прочности цилиндрической панели. Используется математическая модель в виде функционала полной потенциальной энергии деформации. Также приводятся выражения для деформаций, усилий и моментов. Расчетная программа реализована в среде MatLab. Алгоритм построен на методах Ритца и Бубнова – Галёркина. Найдены значения критических нагрузок для различных материалов, получена зависимость прогиба от нагрузки, напряжений от нагрузки, показано поле прогибов. В работе рассматриваются как изотропный, так и ортотропный материалы.

# ПОСТАНОВКА ЗАДАЧИ

По полученному индивидуальному заданию, необходимо провести расчеты напряженно-деформированного состояния с применением метода Ритца к функционалу полной энергии деформации оболочки. В данном случае, был получен вариант пологой оболочки прямоугольного плана. Необходимо рассмотреть пологую оболочку двоякой кривизны с параметрами Ляме *A*= 1*, B*= 1 и главными кривизнами ( Изображение выглядит как черный, темнота

Автоматически созданное описание,  Изображение выглядит как черный, темнота

Автоматически созданное описание ), где  главные радиусы кривизны оболочки вдоль координат *x, y*.

Изображение выглядит как зарисовка, рисунок, линия, диаграмма

Автоматически созданное описание

Рис.1. Визуальная модель рассматриваемой оболочки

# МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Будем рассматривать геометрически нелинейный вариант модели Тимошенко–Рейснера, которая состоит из трех групп соотношений:

1. связь деформаций и перемещений – геометрические соотношения;
2. связь напряжений и деформаций – физические соотношения;
3. функционал полной потенциальной энергии деформации оболочки.

# ГЕОМЕТРИЧЕСКИЕ СООТНОШЕНИЯ

Для начала рассмотрим геометрические соотношения на срединной поверхности. Углы поворота нормали вдоль направлений α и β соответственно вычисляются так:



Связь деформаций через перемещения (относительное удлинение элемента дуги между рассматриваемыми точками):



Модель Тимошенко–Рейснера связывает срединный и отстоящие от срединного на z слои следующими выражениями:



где , – углы отклонения волокон от нормали.

Деформации для слоя, отстоящего от срединного на z, тогда будут выражаться соотношениями



где – функции изменения кривизн и кручения,



# ФИЗИЧЕСКИЕ СООТНОШЕНИЯ

В направлении осей *Ox* и *Oy* действуют нормальные напряжения связанные с деформациями растяжения сжатия. В плоскостях *xOy* появятся напряжения сдвига  связанное с деформациями сдвига. Физические соотношения связывают компоненты напряжений и компоненты деформации.

При линейно-упругом деформировании оболочек они задаются законом Гука:



# УСИЛИЯ И МОМЕНТЫ

Усилия и моменты получаются интегрированием компонент напряжений по толщине оболочки:



На рисунке 2 представлены усилия и моменты, приведенные к срединной поверхности.

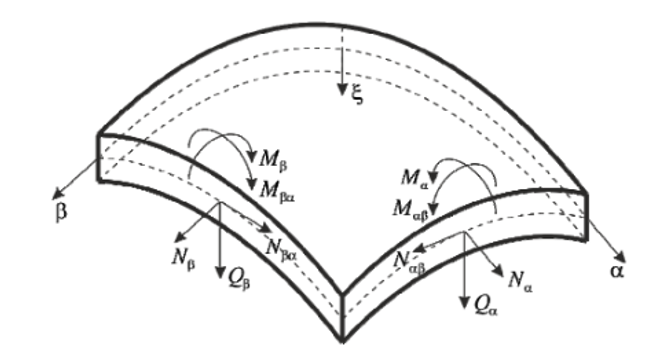


Рис.3 Усилия и моменты, приведенные к срединной поверхности оболочки.

# ФУНКЦИОНАЛ ПОЛНОЙ ЭНЕРГИИ ДЕФОРМАЦИИ

Функционал полной потенциальной энергии деформации представляет собой разность работ внутренних и внешних сил, который при действии внешней равномерно распределенной нагрузки можно записать в виде:



где







Здесь  – компоненты внешней нагрузки вдоль осей *x, y, z.*



Для перехода к локальной системе координат в данном случае потребуется только один угол, соответствующий направлению по оси *у*:



Используем их для вычисления компонент нагрузки от собственного веса:



Также исходя из условия минимума этого функционала при необходимости можно получить уравнения равновесия:







# АЛГОРИТМЫ И МЕТОДЫ

Метод Ритца

Поскольку минимум функционала аналитически найти не представляется возможным, применим один из классических методов минимизации – метод Ритца.  
Минимизирующие функции U, V, W, , будем искать в виде линейной комбинации аппроксимирующих функций:



.

где







Коэффициенты являются неизвестными. Для их поиска необходимо найти градиент функционала и приравнять его к нулю.

Для метода Ритца необходимо иметь систему базисных функций, которые должны удовлетворять граничным условиям, вытекающим из уравнений равновесия, Вводя область , в которой мы рассматриваем поставленную нами задачу, граничные условия можно записать следующим образом:



Поскольку закрепление конструкции было выбрано шарнирно-неподвижное, кроме условий выше, также должны выполняться следующие условия



# Метод Бубнова–Галёркина

Рассматривается краевая задача: найти решение уравнений равновесия (системы линейных дифференциальных уравнений)



где в некоторой области  удовлетворяющее на границе Г области однородным краевым условиям.

Возьмем приближенное решение в виде

# 

Подставляя его в систему, получим невязку



Если  – точное решение уравнения системы, то невязка равна нулю. Если невязка близка к нулю, то можно считать, что она ортогональна к аппроксимирующим функциям. Условие ортогональности имеет вид



После вычисления интегралов получим систему алгебраических уравнений, решив которую найдем неизвестные коэффициенты. Подставив найденные коэффициенты получим приближенное решение краевой задачи.

#### Метод Ньютона для решения СНАУ

Полученная система является нелинейной алгебраической системой от 5N переменных. В общем случае аналитически такая система не решается, поэтому для поиска неизвестных был выбран метод Ньютона для СНАУ.

Обозначим за набор искомых переменных, а за нулевое приближение. Тогда каждое последующее приближение будет выражаться через предыдущее по формуле



где – матрица Гессе функционала 

Задав необходимую точность , процесс вычисления приближений можно остановить при удовлетворении условия



В данной работе при проведении расчетов рассматривается последовательное нагружение панели, поэтому после нахождения решения системы для нагрузки  за нулевое приближение  для следующего нагружения принимается решение .



# АНАЛИЗ ПРОЧНОСТИ ОБОЛОЧКИ

Прочность конструкций, подверженных воздействию внешних нагрузок, анализируется по предельному состоянию материалов, из которых эти конструкции изготовлены.

Под потерей прочности здесь понимается состояние, когда в материале начинают происходить необратимые изменения. Фактически, это определяется, если хотя бы в одной точке конструкции перестает выполняться критерий прочности.

При дальнейшем увеличении нагрузки области невыполнения условий прочности начинают увеличиваться, и анализ их распределения и развития является актуальной задачей.

# КРИТЕРИЙ ПРЕДЕЛЬНОГО СОСТОЯНИЯ ДЛЯ ИЗОТРОВНОГО МАТЕРИАЛА

Для изотропных материалов основные критерии прочности были разработаны еще в конце 19 века (Г. Галилей, Г. Ламе, А. Клебша, Ф. Мариотт, М. Навье, Ш. О. Кулон, И. Баушингер, И. Максвелл, М. Т. Губер, Р. Мизес, Н. Генки и др.).

Суть критериев предельного состояния материала состоит в выборе некоторой функции компонент тензора напряжений, сохраняющей свое значение при напряжениях, соответствующих предельному состоянию материала, независимо от соотношений компонент тензора напряжений, при которых производится нагружение.

Константы материала *F* находятся по результатам испытаний, которые проводятся при одноосном растяжении или сжатии и кручении. Число этих констант для разных критериев различно. Для изотропных материалов это число обычно не превышает трех.

Геометрически условие прочности представляет собой некоторую предельную поверхность, которая ограничивает область безопасных состояний материала конструкции.

# КРИТЕРИЙ МИЗЕСА

В данной курсовой работе в качестве критерия предельного состояния для изотропного материала используется критерий Мизеса.



Здесь  – предельные значения напряжения при растяжении, а – главные нормальные напряжения.

Критерий применительно к исследованию прочности оболочечных конструкций записывается в виде:



где – предел текучести материала, – интенсивность напряжений, которую можно представить в виде



.

# КРИТЕРИЙ ПРЕДЕЛЬНОГО СОСТОЯНИЯ ДЛЯ ОРТОТРОПНОГО МАТЕРИАЛА

Из анализа критериев прочности, применимых для ортотропных материалов, следует, что не существует одного универсального, поэтому приходится использовать несколько критериев с последующим анализом полученных результатов.

В исследованиях, посвященных прочности материалов и конструкций, достаточно часто встречаются: критерий максимальных напряжений, критерий Гольденблата – Копнова, критерий Цая – Ву, критерий Хоффмана, критерий Хашина.

В данной курсовой работе в качестве критерия предельного состояния для ортотропного материала используется критерий Цая – Ву, который выглядит так:



# 

# МЕТОДЫ УСКОРЕНИЯ АЛГОРИТМА

.

# ВХОДНЫЕ ДАННЫЕ

Для автоматизации расчетов была написана программа в пакете прикладных программ для решения задач технических вычислений («MATLAB R2021b»). В её основе использовался вышеописанный алгоритм.

Для наглядного представления информации был создан графики (рисунок 2– рисунок 13) при различных значения M space straight и space N для линейной задачи.

В работе использовались нагрузка, изменяющаяся по параболическому закону с возрастанием к центру по оси *Ox,* и равномерно распределенная нагрузка по оси *Oy*.

Исследования проводились на материалах Сталь С345 и Углепластик Т300/976.Для обоих материалов рассматривались волокна под углом в 90 градусов.

В качестве критерия для изотропного материала был применен критерий Мизеса, для ортотропного материала – Цая-Ву. Закрепление было шарнирно-неподвижным. В ходе работы было рассмотрено различное количество аппроксимирующих функций.

Остальные входные данные (в том числе характеристики материалов) приведены в таблице 1.

Таблица 1. Численные данные

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Обоз. | Определение | Ед.  измер. | Значения  для стали | Значения для углепластика |
|  | предельное значения вдоль оси координат *x* | м | 20 | 20 |
|  | предельное значения вдоль оси координат *y* | рад |  |  |
|  | толщина оболочки | м | 0.01 | 0.01 |
|  | главный радиус кривизны оболочки вдоль оси *x* | м |  |  |
|  | главный радиус кривизны оболочки вдоль оси y | м | 5.4 | 5.4 |
|  | модуль упругости | МПа |  |  |
|  | модуль упругости | МПа |  |  |
|  | коэффициент Пуассона | – | 0.3 | 0.29 |
|  | модуль сдвига | МПа |  |  |
|  | модуль сдвига | МПа |  |  |
|  | модуль сдвига | МПа |  |  |
|  | предел прочности для ортотропного материала | МПа | – | 1517 |
|  | предел прочности для ортотропного материала | МПа | – | -1599 |
|  | предел прочности для ортотропного материала | МПа | – | 46 |
|  | предел прочности для ортотропного материала | МПа | – | -253 |
|  | предел прочности для ортотропного материала | МПа | – | 41.4 |
|  | предел прочности для ортотропного материала | МПа | – | – |
|  | предел прочности для ортотропного материала | МПа | – | – |
|  | предел текучести для изотропного материала | МПа | 345 | – |
|  | плотность материала |  | 7800 | 1500 |
|  | главная кривизна оболочки вдоль оси *x* | 1/м |  |  |
|  | главная кривизна оболочки вдоль оси *y* | 1/м |  |  |
|  | составляющие поперечной нагрузки, действующие в  направлениях осей координат *x* | МПа | 0 | 0 |
|  | составляющие поперечной нагрузки, действующие в  направлениях осей координат *y* | МПа | 0 | 0 |
|  | параметр Ляме | – | 1 | 1 |
|  | параметр Ляме | – |  |  |
|  | составляющие поперечной нагрузки, действующие в  направлениях осей координат *z* | МПа | 1.5 | 1.5 |
|  | ускорение свободного падения |  | 9.78 | 9.78 |
|  | коэффициент нагрузки по оси х |  | 0 | 0 |
|  | коэффициент нагрузки по оси х |  | 0 | 0 |
|  | коэффициент нагрузки по оси х |  |  |  |
|  | коэффициент нагрузки по оси х |  |  |  |
|  | коэффициент нагрузки по оси у |  | 1 | 1 |
|  | коэффициент нагрузки по оси у |  | 0 | 0 |
|  | коэффициент нагрузки по оси у |  | 0 | 0 |

# РАСЧЕТЫ

# График зависимости q-W для разных толщин материала Сталь С345

В данной главе приведены графики, полученные при расчете для N = 3x3.

Метод Ритца:

Метод Бубнова-Галеркина:

# График зависимости q-W для разных толщин материала Оргстекло

В данной главе приведены графики, полученные при расчете для N = 3x3.

Метод Ритца:

Метод Бубнова-Галеркина:

# График зависимости q-W для разных толщин материала Т300/976

В данной главе приведены графики, полученные при расчете для N = 3x3.

Метод Ритца:

Метод Бубнова-Галеркина:

# Визуализация пологой оболочки

# Визуализация прогибов под воздействием нагрузки

# ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены тонкостенные цилиндрические оболочечные конструкции, состоящие из ортотропного и изотропного материалов и имеющие шарнирно-неподвижное закрепление.

Рассматриваемые конструкции находились под воздействием равномерно распределенной и возрастающей к центру по параболическому закону поперечной нагрузки. Проводилось комплексное исследование оболочки на устойчивость и прочность. Была разработана расчетная программа в среде MatLab, которая позволяет проводить исследования поведения оболочечных конструкций.

Для панелей оболочек, выполненных из ортотропного материала T300/976 и изотропного Сталь С345, были найдены критические нагрузки потери устойчивости, показаны зависимости прогиба от нагрузки, а также зависимости напряжения материала от действующей нагрузки.

После проведения всех исследований можно сделать вывод о том, что материал Углепластик Т300/796 прочнее материала Сталь С345. Критическая нагрузка для ортотропного материала составила 0.4 МПа, а для изотропного материала 0.8 МПа. Оболочка из углепластика ощутимо легче, чем такая же оболочка из стали.

Выяснилось, что при увеличении длины оболочки увеличивается прогиб оболочки.

# СПИСОК ЛИТЕРАТУРЫ

1. Карпов В. В. Математическое моделирование, алгоритмы исследования модели, вычислительный эксперимент в теории оболочек / В. В. Карпов; СПбГАСУ. – СПб., 2006. – 330 с.
2. Карпов В. В. Прочность и устойчивость подкрепленных оболочек вращения: в 2-х ч. Модели и алгоритмы исследования прочности и устойчивости подкрепленных оболочек вращения / В. В. Карпов. – М.: Физматлит, 2010. – Ч. 1. – 288 с.
3. Математическое моделирование: метод. указания / сост.: В. В. Карпов, А. Ю. Сальников, А. А. Семенов; СПбГАСУ. – СПб., 2019. – 42 с.

# ПРИЛОЖЕНИЕ

import math  
import os  
from decimal import \*  
from sympy import \*  
import sympy as sym  
import numpy as np  
import math as m  
import matplotlib.pyplot as plt  
from sympy import factor\_terms  
import time  
import multiprocessing as mp  
import mpmath as vpa  
  
#m to mm  
# Data for programm  
One\_point = 0  
Multi = 1  
A\_Numeric = 1  
Change = 3  
Type\_Resolve = 1# 0 - Ritz 1 - Nuton 2 - Bubnov  
# Data for algorithm  
Start\_w0 = 0.0035  
eps = 0.1  
N\_x = 1  
N\_y = 1  
N = N\_x \* N\_y  
q\_find = 30/3  
Q\_Count\_Steps = 70  
Q\_step = q\_find/Q\_Count\_Steps  
Count\_Es = 12  
# graph points  
Size = 30  
Size\_visualization = 10  
# Data about object main  
H\_coef = 1  
h = 0.22 \* H\_coef #mm  
A\_lenght\_x = 200  
B\_lenght\_y = 200  
R\_1\_x = 5000  
R\_2\_y = 3330  
A = 1  
B = 1  
K\_x = 1 / R\_1\_x  
K\_y = 1 / R\_2\_y  
z = 0  
# Data material  
V = 200 \* 200 \* 0.22 # m^3  
P1 = 1500  
P2 = 7800  
P3 = 1190  
Square = 20 \* 20  
k = 5 / 6  
# Symbols  
Xx = Symbol('x')  
Yy = Symbol('y')  
Hh = Symbol('h')  
# Static\_DATA for variants:  
# Ort:  
E1 = 0  
E2 = 0  
nu\_12 = 0  
nu\_21 = 0  
G12 = 0  
G13 = 0  
G23 = 0  
F1\_min = 0  
F1\_max = 0  
F2\_min = 0  
F2\_max = 0  
Density = 0  
Sigma\_t = 0  
  
if Change == 1:  
 # T300/976  
 E1 = 140000000  
 E2 = 9700000  
 nu\_12 = 0.29  
 nu\_21 = (E2 \* nu\_12) / E1  
 G12 = 5500000  
 G13 = G12  
 G23 = 3300000  
 F1\_min = -1599000  
 F1\_max = 1517000  
 F2\_min = -253000  
 F2\_max\_T300 = 46000  
 F2\_max = 41400  
 Density = 1500  
 Sigma\_t = 100000  
if Change == 2:  
 # Izo:  
 # Org\_Glass\_1:  
 E1 = 3000000  
 E2 = E1  
 nu\_12 = 0.35  
 nu\_21 = nu\_12  
 G12 = 1200000  
 G13 = G12  
 G23 = G12  
 Density = 1190  
 Sigma\_t = 75000  
if Change == 3:  
 # Still:  
 E1 = 210000000  
 E2 = E1  
 nu\_12 = 0.3  
 nu\_21 = nu\_12  
 G12 = 80700000  
 G13 = G12  
 G23 = G12  
 Density = 7800  
 Sigma\_t = 265000  
  
# Settings to integral  
Start\_integral = 0  
def intagrete\_Es(fun,queue,i):  
 timer = time.time()  
 print("Start Intagrate = ",i)  
 Result = fun  
 Result = factor\_terms(Result)  
 Result = nsimplify(Result, tolerance=1e-15).evalf(15)  
 Result = integrate(Result, (Xx, Start\_integral, A\_lenght\_x))  
 Result = factor\_terms(Result)  
 Result = nsimplify(Result, tolerance=1e-15).evalf(15)  
 Result = (1 / 2) \* integrate(Result, (Yy, Start\_integral, B\_lenght\_y))  
 Result = nsimplify(Result, tolerance=1e-15).evalf(15)  
 queue.put(Result)  
 spend\_time = time.time() - timer  
 print("Result[",i," By time = ",spend\_time,"=",Result,)  
 return Result  
def intagrete\_Es\_2(data):  
 fun = data[0]  
 i = data[1]  
 #timer = time.time()  
 #print("Start Intagrate = ",i)  
 Result = fun  
 Result = factor\_terms(Result)  
 #Result = nsimplify(Result, tolerance=1e-15).evalf(15)  
 Result = integrate(Result, (Xx, Start\_integral, A\_lenght\_x))  
 Result = factor\_terms(Result)  
 #Result = nsimplify(Result, tolerance=1e-15).evalf(15)  
 Result = (1 / 2) \* integrate(Result, (Yy, Start\_integral, B\_lenght\_y))  
 #Result = nsimplify(Result, tolerance=1e-15).evalf(15)  
 #queue.put(Result)  
 #spend\_time = time.time() - timer  
 #print("Result[",i," By time = ",spend\_time,"=",Result,)  
  
 Result = Result.expand()  
 timer = time.time()  
 Result = factor\_terms(Result)  
  
 return Result  
def intagrete\_Es\_Simp(fun,queue,i):  
 timer = time.time()  
 print("Start Intagrate = ", i)  
 Result = fun  
 #Result = factor\_terms(Result)  
  
 Intervals\_count = 8  
  
 h1 = A\_lenght\_x/Intervals\_count  
 h2 = B\_lenght\_y/Intervals\_count  
  
 integrate\_x = 0  
 integrate\_x += Result.subs(Xx,0)  
 integrate\_x += Result.subs(Xx,A\_lenght\_x)  
 integrate\_x = integrate\_x.evalf()  
  
 for Index in range(1, Intervals\_count):  
 if Index % 2:  
 integrate\_x += 4 \* Result.subs(Xx, Index \* h1)  
 else:  
 integrate\_x += 2 \* Result.subs(Xx, Index \* h1)  
 integrate\_x \*= h1/3  
 Buf\_function = integrate\_x.evalf()  
  
 #Buf\_function =  
 #Result = integrate\_x  
 #Result = nsimplify(integrate\_x, tolerance=1e-15).evalf(15)  
 Buf\_function = factor\_terms(Buf\_function)  
 integrate\_y = 0  
 integrate\_y += Buf\_function.subs(Yy, 0)  
 integrate\_y += Buf\_function.subs(Yy, B\_lenght\_y)  
 integrate\_y = integrate\_y.evalf()  
  
 for Index in range(1,Intervals\_count):  
 if Index % 2:  
 integrate\_y += 4 \* Buf\_function.subs(Yy, Index \* h2)  
 else:  
 integrate\_y += 2 \* Buf\_function.subs(Yy, Index \* h2)  
  
 integrate\_y \*= h2/3  
  
 Result = integrate\_y.evalf()  
 #Result = factor\_terms(Result)  
 #Result = nsimplify(Result, tolerance=1e-10).evalf(10)  
 if N == 1:  
 Result = Result.simplify()  
 getcontext().prec = 6  
 #Result = Decimal(Result)  
 #Result = nsimplify(Result, tolerance=1e-22).evalf(22)  
 queue.put(Result)  
 spend\_time = time.time() - timer  
 #print("Result[",i," By time = ",spend\_time,"=",Result,)  
 print("Result[", i, " By time = ", spend\_time )  
def intagrete\_Es\_Simp\_2(data):  
 fun = data[0]  
 i = data[1]  
 timer = time.time()  
 #print("Start Intagrate = ", i)  
 Result = fun  
 #Result = factor\_terms(Result)  
  
 Intervals\_count = 5  
  
 h1 = A\_lenght\_x/Intervals\_count  
 h2 = B\_lenght\_y/Intervals\_count  
  
 integrate\_x = 0  
 integrate\_x += Result.subs(Xx,0)  
 integrate\_x += Result.subs(Xx,A\_lenght\_x)  
 integrate\_x = integrate\_x.evalf(15)  
  
 for Index in range(1, Intervals\_count):  
 if Index % 2:  
 integrate\_x += 4 \* Result.subs(Xx, Index \* h1)  
 else:  
 integrate\_x += 2 \* Result.subs(Xx, Index \* h1)  
 integrate\_x \*= h1/3  
 Buf\_function = integrate\_x.evalf(15)  
  
 #Buf\_function =  
 #Result = integrate\_x  
 #Result = nsimplify(integrate\_x, tolerance=1e-15).evalf(15)  
 #Buf\_function = factor\_terms(Buf\_function)  
 integrate\_y = 0  
 integrate\_y += Buf\_function.subs(Yy, 0)  
 integrate\_y += Buf\_function.subs(Yy, B\_lenght\_y)  
 integrate\_y = integrate\_y.evalf(15)  
  
 for Index in range(1,Intervals\_count):  
 if Index % 2:  
 integrate\_y += 4 \* Buf\_function.subs(Yy, Index \* h2)  
 else:  
 integrate\_y += 2 \* Buf\_function.subs(Yy, Index \* h2)  
  
 integrate\_y \*= h2/3  
  
 Result = integrate\_y.evalf(15)  
 #Result = factor\_terms(Result)  
 #Result = nsimplify(Result, tolerance=1e-10).evalf(10)  
 #if N == 1:  
 #Result = Result.simplify()  
 #Result = Result.simplify()  
 #mp.dps = 5  
 vpa.mp.dps = 8  
 #Result = vpa.mpf(Result)  
 #Result = trigsimp(Result)  
 #Result = Decimal(Result)  
 #Result = nsimplify(Result, tolerance=1e-22).evalf(22)  
 #queue.put(Result)  
 Result = Result.expand()  
 #timer = time.time()  
 Result = factor\_terms(Result)  
 #Result = Result.simplify()  
 #print("Simplify = By time = ", time.time() - timer )  
 #spend\_time = time.time() - timer  
 #print("Result[",i," By time = ",spend\_time,"=",Result,)  
 #print("Result[", i, "] By time = ", spend\_time )  
 return Result  
def get\_diff(fun,queue,symbol,i):  
 timer = time.time()  
 print("start diff =",i)  
 Result = []  
 Result.append(fun.diff(symbol))  
 Result.append(i)  
 queue.put(Result)  
 spend\_time = time.time() - timer  
 print("Result[", i, " By time = ", spend\_time)  
 return 0  
def get\_diff\_2(data):  
 i = data[2]  
 symbol = data[1]  
 fun = data[0]  
 timer = time.time()  
 #print("start diff =",i)  
 Result = []  
 Result = fun.diff(symbol)  
 #Result.append(i)  
 #queue.put(Result)  
 spend\_time = time.time() - timer  
 #print("Result[", i, " By time = ", spend\_time)  
 return Result  
def get\_diff\_2\_Ja(data):  
 i = data[2]  
 symbol = data[1]  
 fun = data[0]  
 timer = time.time()  
 #print("start diff =",i)  
 Result = []  
 buf\_result = 0  
 for index in range(0,Count\_Es):  
 buf\_result+=fun[index].diff(symbol)  
 Result = buf\_result  
 #Result.append(i)  
 #queue.put(Result)  
 spend\_time = time.time() - timer  
 #print("Result[", i, " By time = ", spend\_time)  
 return Result  
  
def get\_diff\_2arg(fun,queue,symbol,i,j):  
 timer = time.time()  
 print("start diff =",i,",",j)  
 Result = []  
 Result.append(fun.diff(symbol))  
 Result.append(i)  
 Result.append(j)  
 queue.put(Result)  
 spend\_time = time.time() - timer  
 print("Result[", i,",",j, " By time = ", spend\_time)  
 return 0  
def get\_diff\_2arg\_2(data):  
 fun = data[0]  
 symbol = data[1]  
 i = data[2]  
 j = data[3]  
 #print(i, j)  
 timer = time.time()  
 #print("start diff =",i,",",j)  
 Result = []  
 Result.append(fun.diff(symbol))  
 Result.append((i,j))  
 #Result.append(j)  
 #queue.put(Result)  
 spend\_time = time.time() - timer  
 #print("Result[", i,",",j, " By time = ", spend\_time)  
 return Result  
def get\_diff\_2arg\_2\_Ja(data):  
 fun = data[0]  
 symbol = data[1]  
 i = data[2]  
 j = data[3]  
 #print(i, j)  
 timer = time.time()  
 #print("start diff =",i,",",j)  
 Result = []  
 buf\_result = 0  
 for index in range(0,Count\_Es):  
 buf\_result+=fun[index].diff(symbol)  
 Result.append(buf\_result)  
 Result.append((i,j))  
 #Result.append(j)  
 #queue.put(Result)  
 spend\_time = time.time() - timer  
 #print("Result[", i,",",j, " By time = ", spend\_time)  
 return Result  
  
def Sub\_data(data):  
 fun = data[0]  
 Symbols\_get = data[1]  
 #print("Sym\_data =",Symbols\_get)  
 Result = fun.subs(Symbols\_get)  
 return Result  
def intagrete\_Es\_Simp\_Bubnov(fun, queue, i):  
 timer = time.time()  
 print("Start Intagrate = ", i)  
 Result = fun  
 Result = factor\_terms(Result)  
  
 Intervals\_count = 10  
  
 h1 = A\_lenght\_x / Intervals\_count  
 h2 = B\_lenght\_y / Intervals\_count  
  
 integrate\_x = 0  
 integrate\_x += Result.subs(Xx, 0)  
 integrate\_x += Result.subs(Xx, A\_lenght\_x)  
 integrate\_x = integrate\_x.evalf()  
  
 for Index in range(1, Intervals\_count):  
 if Index % 2:  
 integrate\_x += 4 \* Result.subs(Xx, Index \* h1)  
 else:  
 integrate\_x += 2 \* Result.subs(Xx, Index \* h1)  
 integrate\_x \*= h1 / 3  
 Buf\_function = integrate\_x.evalf()  
  
 # Buf\_function =  
 # Result = integrate\_x  
 # Result = nsimplify(integrate\_x, tolerance=1e-15).evalf(15)  
 Buf\_function = factor\_terms(Buf\_function)  
 integrate\_y = 0  
 integrate\_y += Buf\_function.subs(Yy, 0)  
 integrate\_y += Buf\_function.subs(Yy, B\_lenght\_y)  
 integrate\_y = integrate\_y.evalf()  
  
 for Index in range(1, Intervals\_count):  
 if Index % 2:  
 integrate\_y += 4 \* Buf\_function.subs(Yy, Index \* h2)  
 else:  
 integrate\_y += 2 \* Buf\_function.subs(Yy, Index \* h2)  
  
 integrate\_y \*= h2 / 3  
  
 Result = integrate\_y.evalf()  
 # Result = factor\_terms(Result)  
 # Result = nsimplify(Result, tolerance=1e-15).evalf(15)  
 Result = Result.simplify()  
 Result = nsimplify(Result, tolerance=1e-22).evalf(22)  
 return\_val = [0]\*2  
 return\_val[0] = Result  
 return\_val[1] = i  
 queue.put(return\_val)  
 spend\_time = time.time() - timer  
 # print("Result[",i," By time = ",spend\_time,"=",Result,)  
 print("Result[", i, " By time = ", spend\_time)  
def Get\_W\_Plane(x\_val, y\_val, function, Values, type):  
 W\_result = 0  
  
 # Hard  
 if type == 1:  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 W\_result += Values[(j - 1) \* N\_x + i - 1] \* sin(2 \* i \* x\_val \* m.pi / A\_lenght\_x) \* sin(  
 (2 \* j - 1) \* y\_val \* m.pi / B\_lenght\_y)  
 if type == 2:  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 W\_result += Values[(j - 1) \* N\_x + i - 1] \* sin((2 \* i - 1) \* x\_val \* m.pi / A\_lenght\_x) \* sin(  
 2 \* j \* y\_val \* m.pi / B\_lenght\_y)  
 if type == 3:  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 W\_result += Values[(j - 1) \* N\_x + i - 1] \* sin((2 \* i - 1) \* x\_val \* m.pi / A\_lenght\_x) \* sin(  
 (2 \* j - 1) \* y\_val \* m.pi / B\_lenght\_y)  
  
 return W\_result  
def Draw\_deformed\_plot(W\_func,U\_func,V\_func):  
 print("Start Visual!!!")  
 x\_array = [0] \* (Size\_visualization\*Size\_visualization)  
 y\_array = [0] \* (Size\_visualization\*Size\_visualization)  
 x\_array\_graph = [0] \* (Size\_visualization\*Size\_visualization)  
 y\_array\_graph = [0] \* (Size\_visualization\*Size\_visualization)  
 pi = 3.14  
  
 Min\_ab = min(A\_lenght\_x,B\_lenght\_y)  
 Max\_ab = max(A\_lenght\_x,B\_lenght\_y)  
 #print("Min ab = ",Min\_ab)  
 #print("Max ab = ", Min\_ab)  
 Min\_ab\_g = min(pi \* R\_1\_x, pi \* R\_2\_y)  
 Max\_ab\_g = max(pi \* R\_1\_x, pi \* R\_2\_y)  
 #print("Min ab\_g = ", Min\_ab\_g)  
 #print("Max ab\_g = ", Max\_ab\_g)  
  
 X\_array = [0] \* (Size\_visualization \* Size\_visualization)  
 Y\_array = [0] \* (Size\_visualization \* Size\_visualization)  
 Z\_array = [] \* (Size\_visualization \* Size\_visualization)  
  
 R\_main = max(R\_1\_x,R\_2\_y) - min(R\_1\_x,R\_2\_y)  
 r\_main = min(R\_1\_x,R\_2\_y)  
  
 #print("R\_main = ", R\_main)  
 #print("r\_main = ", r\_main)  
  
 #x\_step = (2\*Min\_ab / (2 \* r\_main))  
 #y\_step = (2\*Max\_ab / (2 \* (R\_main + r\_main)))  
 x\_step = A\_lenght\_x/(Size\_visualization\*Size\_visualization)  
 y\_step = B\_lenght\_y / (Size\_visualization\*Size\_visualization)  
  
 #print("X\_step =",x\_step)  
 #print("Y\_step = ",y\_step)  
  
 x\_array[0] = 0  
 y\_array[0] = 0  
 x\_array\_graph[0] = 0  
 y\_array\_graph[0] = 0  
 x\_graph\_step = (2\*Min\_ab\_g / (2 \* r\_main)) / (Size\_visualization\*\*2)  
 y\_graph\_step = (2\*Max\_ab\_g / (2 \* (R\_main + r\_main))) / (Size\_visualization \*\* 2)  
  
 for index in range(1,Size\_visualization\*Size\_visualization):  
 x\_array[index] = x\_array[index - 1] + x\_step  
 y\_array[index] = y\_array[index - 1] + y\_step  
 x\_array\_graph[index] = x\_array\_graph[index - 1] + x\_graph\_step  
 y\_array\_graph[index] = y\_array\_graph[index - 1] + y\_graph\_step  
  
  
 #print("X array",x\_array)  
 #print("Y array",y\_array)  
  
 for index in range(0,Size\_visualization\*\*2):  
 buf = [0] \* (Size\_visualization\*\*2)  
 Z\_array.append(buf)  
 for index\_2 in range(0,Size\_visualization\*\*2):  
 Z\_array[index][index\_2] = (((r\_main\*sin(x\_array\_graph[index\_2]) + r\_main\*sin(y\_array\_graph[index])))\*h)/1000  
 #print("Z0 = " ,Z\_array[0])  
 #print("Z\_len", len(Z\_array))  
 print("H\_max =", max(max(Z\_array)))  
 Z = np.array(Z\_array)  
 x\_array\_graph = np.array(x\_array\_graph)  
 y\_array\_graph = np.array(y\_array\_graph)  
 X, Y = np.meshgrid(x\_array, y\_array)  
  
 #print("X[0][0]",X[0][0])  
 #for i in range(0,len(X)):  
 #print(X[i])  
  
 fig = plt.figure()  
 #ax = plt.axes(projection='3d',ylim =(0,200),xlim = (0,200),zlim = (0,200))  
 ax = plt.axes(projection='3d', ylim=(0, 200), xlim=(0, 200))  
 ax.plot\_surface(X, Y, Z, cmap='viridis', edgecolor='green')  
 ax.set\_xlabel('x [мм]')  
 ax.set\_ylabel('y [мм]')  
 ax.set\_zlabel('z [мм]')  
 ax.set\_title("Visual")  
 plt.show()  
  
 Nu\_array = [0] \* 3  
 Nv\_array = [0] \* 3  
 Nw\_array = [0] \* 3  
  
 Nu\_array[0] = -sin(Xx)\*sin(Yy)\*U\_func  
 Nu\_array[1] = -sin(Xx)\*cos(Yy)\*U\_func  
 Nu\_array[2] = cos(Xx)\*V\_func  
  
 Nv\_array[0] = cos(Yy) \* V\_func  
 Nv\_array[1] = -sin(Yy) \* V\_func  
 Nv\_array[2] = 0 \* V\_func  
  
 Nw\_array[0] = cos(Xx)\*sin(Yy) \* W\_func  
 Nw\_array[1] = cos(Xx)\*cos(Yy) \* W\_func  
 Nw\_array[2] = sin(Xx) \* W\_func  
  
 print(W\_func)  
 print(U\_func)  
 print(V\_func)  
 print(x\_array)  
  
 x\_new = np.linspace(0,A\_lenght\_x,Size\_visualization\*Size\_visualization)  
 y\_new = np.linspace(0,B\_lenght\_y,Size\_visualization\*Size\_visualization)  
  
 for index in range(0,Size\_visualization\*Size\_visualization):  
 for index\_2 in range(0, Size\_visualization \* Size\_visualization):  
 #X[index][index\_2] = X[index][index\_2] + Nu\_array[0].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])]) \  
 # + Nv\_array[0].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])]) + Nw\_array[0].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])])  
 #Y[index][index\_2] = Y[index][index\_2] + Nu\_array[1].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])]) \  
 # + Nv\_array[2].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])]) + Nw\_array[2].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])])  
 #Z\_array[index][index\_2] = Z\_array[index][index\_2]/10 + Nw\_array[1].subs([(Xx, x\_new[index\_2]), (Yy, y\_new[index])]) \  
 # + Nw\_array[1].subs([(Xx, x\_new[index\_2]), (Yy, y\_new[index])]) + Nw\_array[2].subs([(Xx, x\_new[index\_2]), (Yy, y\_new[index])])  
 X[index][index\_2] = X[index][index\_2] - U\_func.subs([(Xx, x\_new[index]), (Yy, y\_new[index\_2])])  
 Y[index][index\_2] = Y[index][index\_2] - V\_func.subs( [(Xx, x\_new[index]), (Yy, y\_new[index\_2])])  
 Z\_array[index][index\_2] =Z\_array[index][index\_2] - W\_func.subs( [(Xx, x\_new[index]), (Yy, y\_new[index\_2])])\*10  
 #print("old",Z\_array[index][index\_2]/1000)  
 #print("new",Nu\_array[2].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])]) \  
 # + Nv\_array[2].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])]) + Nw\_array[2].subs([(Xx, x\_array[index\_2]), (Yy, y\_array[index])]))  
  
 #z\_array = np.array(d\_z\_array)  
 #x\_array\_graph = np.array(x\_array\_graph)  
 #y\_array\_graph = np.array(y\_array\_graph)  
 #X, Y = np.meshgrid(x\_array\_graph, y\_array\_graph)  
 Z = np.array(Z\_array)  
  
 fig = plt.figure()  
 ax = plt.axes(projection='3d')  
 ax.plot\_surface(X, Y, Z, cmap='viridis', edgecolor='green')  
 ax.set\_xlabel('x [мм]')  
 ax.set\_ylabel('y [мм]')  
 ax.set\_zlabel('z [мм]')  
 ax.set\_title("Visual")  
 plt.show()  
  
 fig = plt.figure()  
 ax = plt.axes(projection='3d', ylim=(0, 200), xlim=(0, 200),zlim = (0,200))  
 ax.plot\_surface(X, Y, Z, cmap='viridis', edgecolor='green')  
 ax.set\_xlabel('x [мм]')  
 ax.set\_ylabel('y [мм]')  
 ax.set\_zlabel('z [мм]')  
 ax.set\_title("Visual")  
 plt.show()  
  
def Draw\_3d\_W(Function, Values\_Result, type\_f):  
 x\_array = []  
 y\_array = []  
 z\_array = [0] \* Size  
 Max\_Value = 0  
 Get\_W = Function.copy()  
 # print(Values\_Result)  
 print(Function)  
 # print()  
 # print(Function)  
  
 for i in range(0, Size):  
 z\_array[i] = [0] \* Size  
  
 step\_x = A\_lenght\_x / (Size - 1)  
 step\_y = B\_lenght\_y / (Size - 1)  
  
 for i in range(0, Size):  
 x\_array.append(i \* step\_x)  
 y\_array.append(i \* step\_y)  
  
 for j in range(0, Size):  
 for i in range(0, Size):  
 z\_array[i][j] = Get\_W.subs( [(Xx, x\_array[i]), (Yy, y\_array[j])])  
  
 z\_array = np.array(z\_array)  
 x\_array = np.array(x\_array)  
 y\_array = np.array(y\_array)  
 X, Y = np.meshgrid(x\_array, y\_array)  
  
 fig = plt.figure()  
 ax = plt.axes(projection='3d')  
 ax.plot\_surface(X, Y, z\_array, cmap='viridis', edgecolor='green')  
 ax.set\_xlabel('x [м]')  
 ax.set\_ylabel('y [м]')  
 ax.set\_zlabel("W" + ' (x,y) [м]')  
 # ax.set\_title(Function + ' (x,y)')  
 plt.show()  
 return Max\_Value  
def Draw\_3d\_Sigmas\_main(Function, Values\_Result, Type\_Sigmas, U\_function, V\_function, W\_Function,PsiX\_function, PsiY\_function, z\_val, W\_val, U\_val,V\_val, PsiX\_val,PsiY\_val,Sigma\_x,Sigma\_y,Sigma\_tay):  
 x\_array = []  
 y\_array = []  
 z\_array = [0] \* Size  
 Max\_Value = 0  
  
 for i in range(0, Size):  
 z\_array[i] = [0] \* Size  
  
 step\_x = A\_lenght\_x / (Size - 1)  
 step\_y = B\_lenght\_y / (Size - 1)  
  
 for i in range(0, Size):  
 x\_array.append(i \* step\_x)  
 y\_array.append(i \* step\_y)  
  
 Sigma\_x\_Function = 5 \* Xx  
 Sigma\_y\_Function = 5 \* Xx  
 Tay\_xy\_Function = 5 \* Xx  
 Max\_Sigmas\_values = [0]\*3  
  
 if Change == 1:  
 Sigma\_x\_Function = (Get\_Sigma\_x\_Orto(U\_function,V\_function,W\_Function,E1,nu\_12, nu\_21,z\_val))  
 Sigma\_y\_Function = (Get\_Sigma\_y\_Orto(U\_function,V\_function,W\_Function,E2,nu\_12, nu\_21,z\_val))  
 Tay\_xy\_Function = (Get\_Sigma\_tay\_Orto(U\_function,V\_function,W\_Function,G12,z\_val))  
 if Change == 2:  
 Sigma\_x\_Function = Get\_Sigma\_x\_Izo(U\_function, V\_function, W\_Function, PsiX\_function, PsiY\_function, E1, nu\_12,z\_val)  
 Sigma\_y\_Function = Get\_Sigma\_y\_Izo(U\_function, V\_function, W\_Function, PsiX\_function, PsiY\_function, E1, nu\_12,z\_val)  
 Tay\_xy\_Function = Get\_Sigma\_tay\_Izo(U\_function, V\_function, W\_Function, PsiX\_function, PsiY\_function, E1, nu\_12,z\_val)  
 if Change == 3:  
 #Sigma\_x, Sigma\_y, Sigma\_tay  
 #Sigma\_x\_Function = (Get\_Sigma\_x\_Izo(U\_function, V\_function, W\_Function, PsiX\_function, PsiY\_function, E1\_Still, nu\_Still, z\_val))  
 Sigma\_x\_Function = Sigma\_x.copy()  
 #Sigma\_y\_Function = (Get\_Sigma\_y\_Izo(U\_function, V\_function, W\_Function, PsiX\_function, PsiY\_function, E1\_Still, nu\_Still, z\_val))  
 Sigma\_y\_Function = Sigma\_y.copy()  
 #Tay\_xy\_Function = (Get\_Sigma\_tay\_Izo(U\_function, V\_function, W\_Function, PsiX\_function, PsiY\_function, E1\_Still, nu\_Still, z\_val))  
 Tay\_xy\_Function = Sigma\_tay.copy()  
  
 for i in range(N+1):  
 Sigma\_x\_Function = Sigma\_x\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
 Sigma\_y\_Function = Sigma\_y\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
 Tay\_xy\_Function = Tay\_xy\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
  
 #print(Sigma\_x\_Function)  
 #print(Sigma\_y\_Function)  
 #print(Tay\_xy\_Function)  
  
  
 for j in range(0, Size):  
 for i in range(0, Size):  
 Max\_Sigmas\_values[0] = Sigma\_x\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
 Max\_Sigmas\_values[1] = Sigma\_y\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
 Max\_Sigmas\_values[2] = Tay\_xy\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
  
 z\_array[i][j] =((Max\_Sigmas\_values[0] \*\* 2) + (Max\_Sigmas\_values[1] \*\* 2) - Max\_Sigmas\_values[0] \* Max\_Sigmas\_values[1] + 3 \* (Max\_Sigmas\_values[2] \*\* 2)) \*\* (0.5)  
  
 if abs(z\_array[i][j]) > abs(Max\_Value):  
 Max\_Value = z\_array[i][j]  
  
 #plt.show()  
 print("Max value Miz= ",Max\_Value)  
 Sigma\_Tay\_krit = 150  
 if Change ==3:  
 Sigma\_Tay\_krit = 340  
  
 return Max\_Value/Sigma\_Tay\_krit  
def get\_Mizz(W\_val, U\_val,V\_val, PsiX\_val,PsiY\_val,Sigma\_x,Sigma\_y,Sigma\_tay):  
 x\_array = []  
 y\_array = []  
 z\_array = [0] \* Size  
  
 for i in range(0, Size):  
 z\_array[i] = [0] \* Size  
  
 step\_x = A\_lenght\_x / (Size - 1)  
 step\_y = B\_lenght\_y / (Size - 1)  
  
 for i in range(0, Size):  
 x\_array.append(i \* step\_x)  
 y\_array.append(i \* step\_y)  
  
 Sigma\_x\_Function = 5 \* Xx  
 Sigma\_y\_Function = 5 \* Xx  
 Tay\_xy\_Function = 5 \* Xx  
 Max\_Sigmas\_values = [0]\*3  
  
 if Change == 1:  
 print("Change == 1")  
 if Change == 2:  
 print("Change == 2")  
 if Change == 3:  
 Sigma\_x\_Function = Sigma\_x.copy()  
 Sigma\_y\_Function = Sigma\_y.copy()  
 Tay\_xy\_Function = Sigma\_tay.copy()  
  
 #timer = time.time()  
 for i in range(N+1):  
 Sigma\_x\_Function = Sigma\_x\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
 Sigma\_y\_Function = Sigma\_y\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
 Tay\_xy\_Function = Tay\_xy\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
 #print("Spend time to Mizz =",time.time() - timer)  
 x\_Mizz = 0  
 y\_Mizz = 0  
 Max\_Value = 0  
 Mx\_max = 0  
 My\_max = 0  
 mt\_max = 0  
 for j in range(0, Size):  
 for i in range(0, Size):  
 Max\_Sigmas\_values[0] = Sigma\_x\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
 Max\_Sigmas\_values[1] = Sigma\_y\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
 Max\_Sigmas\_values[2] = Tay\_xy\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
 if abs(Max\_Sigmas\_values[0])>Mx\_max:  
 Mx\_max = abs(Max\_Sigmas\_values[0])  
 if abs(Max\_Sigmas\_values[1])>My\_max:  
 My\_max = abs(Max\_Sigmas\_values[1])  
 if abs(Max\_Sigmas\_values[2])>mt\_max:  
 mt\_max = abs(Max\_Sigmas\_values[2])  
  
 z\_array[i][j] =((Max\_Sigmas\_values[0] \*\* 2) + (Max\_Sigmas\_values[1] \*\* 2) - Max\_Sigmas\_values[0] \* Max\_Sigmas\_values[1] + 3\*(Max\_Sigmas\_values[2] \*\* 2)) \*\* (0.5)  
 #if i == 10 and j == 10:  
 #print("Mizz in center =",z\_array[i][j])  
 if abs(z\_array[i][j]) > abs(Max\_Value):  
 Max\_Value = z\_array[i][j]  
 x\_Mizz = x\_array[i]  
 y\_Mizz = y\_array[j]  
  
 print("Mx =", Mx\_max)  
 print("My =", My\_max)  
 print("Mtay =", mt\_max)  
 #plt.show()  
 #print("Max value Miz= ",Max\_Value)  
 #Sigma\_Tay\_krit = 150  
 #if Change ==3:  
 #Sigma\_t = 300  
  
 return Max\_Value,x\_Mizz,y\_Mizz  
def get\_Mizz\_2(W\_val, U\_val,V\_val, PsiX\_val,PsiY\_val,Sigma\_x,Sigma\_y,Sigma\_tay):  
 x\_array = []  
 y\_array = []  
 z\_array = [0] \* Size  
  
 for i in range(0, Size):  
 z\_array[i] = [0] \* Size  
  
 step\_x = A\_lenght\_x / (Size - 1)  
 step\_y = B\_lenght\_y / (Size - 1)  
  
 for i in range(0, Size):  
 x\_array.append(i \* step\_x)  
 y\_array.append(i \* step\_y)  
  
 Sigma\_x\_Function = 5 \* Xx  
 Sigma\_y\_Function = 5 \* Xx  
 Tay\_xy\_Function = 5 \* Xx  
 Max\_Sigmas\_values = [0]\*3  
  
 if Change == 1:  
 print("Change == 1")  
 if Change == 2:  
 print("Change == 2")  
 if Change == 3:  
 Sigma\_x\_Function = Sigma\_x.copy()  
 Sigma\_y\_Function = Sigma\_y.copy()  
 Tay\_xy\_Function = Sigma\_tay.copy()  
  
 #timer = time.time()  
 for i in range(N+1):  
 Sigma\_x\_Function = Sigma\_x\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
 Sigma\_y\_Function = Sigma\_y\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
 Tay\_xy\_Function = Tay\_xy\_Function.subs(  
 [('w' + str(i), W\_val[i - 1]), ('u' + str(i), U\_val[i - 1]), ('v' + str(i), V\_val[i - 1]),  
 ('PsiX' + str(i), PsiX\_val[i - 1]), ('PsiY' + str(i), PsiY\_val[i - 1])])  
 #print("Spend time to Mizz =",time.time() - timer)  
 x\_Mizz = 0  
 y\_Mizz = 0  
 Max\_Value = 0  
 Mx\_max = 0  
 My\_max = 0  
 mt\_max = 0  
 for j in range(0, Size):  
 for i in range(0, Size):  
 Max\_Sigmas\_values[0] = Sigma\_x\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
 Max\_Sigmas\_values[1] = Sigma\_y\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
 Max\_Sigmas\_values[2] = Tay\_xy\_Function.subs([(Xx, x\_array[i]), (Yy, y\_array[j])])  
 if abs(Max\_Sigmas\_values[0])>Mx\_max:  
 Mx\_max = abs(Max\_Sigmas\_values[0])  
 if abs(Max\_Sigmas\_values[1])>My\_max:  
 My\_max = abs(Max\_Sigmas\_values[1])  
 #print(mt\_max)  
 #print(Max\_Sigmas\_values[2])  
 if abs(Max\_Sigmas\_values[2])>mt\_max:  
 mt\_max = abs(Max\_Sigmas\_values[2])  
  
 z\_array[i][j] =((Max\_Sigmas\_values[0] - Max\_Sigmas\_values[1] )\*\*2 + (Max\_Sigmas\_values[1] - 0) \*\* 2 + (Max\_Sigmas\_values[0]) \*\* 2 + 6\*(Max\_Sigmas\_values[2] \*\* 2)) \*\* (0.5)  
 z\_array[i][j] = z\_array[i][j] \* 1/(2 \*\*(0.5))  
 #if i == 10 and j == 10:  
 #print("Mizz in center =",z\_array[i][j])  
 if z\_array[i][j] > Max\_Value:  
 Max\_Value = z\_array[i][j]  
 x\_Mizz = x\_array[i]  
 y\_Mizz = y\_array[j]  
  
 print("Mx =", Mx\_max)  
 print("My =", My\_max)  
 print("Mtay =", mt\_max)  
 #plt.show()  
 #print("Max value Miz= ",Max\_Value)  
 #Sigma\_Tay\_krit = 150  
 #if Change ==3:  
 #Sigma\_t = 300  
  
 return Max\_Value/Sigma\_t,x\_Mizz,y\_Mizz  
  
def Get\_v\_coefs():  
 w\_coefs = []  
 for i in range(1, N + 1):  
 w\_coefs.append(Symbol('v' + str(i)))  
 return w\_coefs  
def Get\_u\_coefs():  
 w\_coefs = []  
 for i in range(1, N + 1):  
 w\_coefs.append(Symbol('u' + str(i)))  
 return w\_coefs  
def Get\_w\_coefs():  
 w\_coefs = []  
 for i in range(1, N + 1):  
 w\_coefs.append(Symbol('w' + str(i)))  
 return w\_coefs  
def Get\_PsiX\_coefs():  
 w\_coefs = []  
 for i in range(1, N + 1):  
 w\_coefs.append(Symbol('PsiX' + str(i)))  
 return w\_coefs  
def Get\_PsiY\_coefs():  
 w\_coefs = []  
 for i in range(1, N + 1):  
 w\_coefs.append(Symbol('PsiY' + str(i)))  
 return w\_coefs  
def Get\_w\_sin\_x(i):  
 return sin((2 \* i - 1) \* Xx \* m.pi / A\_lenght\_x)  
def Get\_w\_sin\_y(j):  
 return sin((2 \* j - 1) \* Yy \* m.pi / B\_lenght\_y)  
def Get\_w\_sin\_x\_2(i):  
 return sin((2 \* i) \* Xx \* m.pi / A\_lenght\_x)  
def Get\_w\_sin\_y\_2(j):  
 return sin((2 \* j) \* Yy \* m.pi / B\_lenght\_y)  
def Get\_w\_cos\_x(i):  
 return cos((2 \* i - 1) \* Xx \* m.pi / A\_lenght\_x)  
def Get\_w\_cos\_y(j):  
 return cos((2 \* j - 1) \* Yy \* m.pi / B\_lenght\_y)  
def Get\_w\_cos\_x\_2(i):  
 return cos((2 \* i) \* Xx \* m.pi / A\_lenght\_x)  
def Get\_w\_cos\_y\_2(j):  
 return cos((2 \* j) \* Yy \* m.pi / B\_lenght\_y)  
def Get\_U\_function\_vals(u\_vals):  
 Result = 0  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 Result += u\_vals[(j - 1) \* N\_x + i - 1] \* Get\_w\_sin\_x\_2(i) \* Get\_w\_sin\_y(j)  
 return Result  
def Get\_V\_function\_vals(v\_vals):  
 Result = 0  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 Result += v\_vals[(j - 1) \* N\_x + i - 1] \* Get\_w\_sin\_x(i) \* Get\_w\_sin\_y\_2(j)  
 return Result  
def Get\_W\_function\_vals(w\_vals):  
 Result = 0  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 Result += w\_vals[(j - 1) \* N\_x + i - 1] \* Get\_w\_sin\_x(i) \* Get\_w\_sin\_y(j)  
 return Result  
def Get\_PsiX\_function\_vals(PsiX\_vals):  
 Result = 0  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 Result += PsiX\_vals[(j - 1) \* N\_x + i - 1] \* Get\_w\_cos\_x(i) \* Get\_w\_sin\_y(j)  
 return Result  
def Get\_PsiY\_function\_vals(PsiY\_vals):  
 Result = 0  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 Result += PsiY\_vals[(j - 1) \* N\_x + i - 1] \* Get\_w\_sin\_x(i) \* Get\_w\_cos\_y(j)  
 return Result  
def ksi\_1(PsiX\_function, PsiY\_function):  
 if (A\_Numeric):  
 Result = PsiX\_function.diff(Xx)/A  
 else:  
 Result = (1 / A) \* PsiX\_function.diff(Xx) + (1 / (A \* B)) \* A.diff(Yy) \* PsiY\_function  
  
 return Result  
def ksi\_2(PsiX\_function, PsiY\_function):  
 if (A\_Numeric):  
 Result = PsiY\_function.diff(Yy)/B  
 else:  
 Result = (1 / B) \* PsiY\_function.diff(Yy) + (1 / (A \* B)) \* B.diff(Xx) \* PsiX\_function  
  
 return Result  
def ksi\_12(PsiX\_function, PsiY\_function):  
 Result\_buf = (PsiY\_function.diff(Xx)/A + PsiX\_function.diff(Yy)/B)  
 if (A\_Numeric):  
 Result\_3 = 0  
 else:  
 Result\_3 = (1 / (A \* B)) \* (A.diff(Yy) \* PsiX\_function + B.diff(Xx) \* PsiY\_function)  
  
 Result = (Result\_buf - Result\_3) / 2  
 return Result  
def Tetta\_1(W\_function, U\_function):  
 Result = -((1/A) \* W\_function.diff(Xx)+K\_x\*U\_function)  
 return Result  
def Tetta\_2(W\_function, V\_function):  
 Result = -((1/B) \* W\_function.diff(Yy)+K\_y\*V\_function)  
 return Result  
def Q\_x(PsiX\_function,G\_13,TETTA\_1):  
 Result = k \* G\_13 \* Hh \* (PsiX\_function - TETTA\_1)  
 return Result  
def Q\_y(PsiY\_function, G\_23,TETTA\_2):  
 Result = k \* G\_23 \* Hh \* (PsiY\_function - TETTA\_2)  
 return Result  
def e\_x(U\_function, V\_function, W\_Function,TETTA\_1):  
 if (A\_Numeric):  
 Result = (1 / A) \* U\_function.diff(Xx) - K\_x \* W\_Function + (1 / 2) \* (TETTA\_1\*TETTA\_1)  
 else:  
 Result = (1 / A) \* U\_function.diff(Xx) - K\_x \* W\_Function + (1 / 2) \* (TETTA\_1 \*\* 2) - (  
 1 / (A \* B)) \* V\_function \* A.diff(Yy)  
 return Result  
def e\_y(U\_function, V\_function, W\_Function,TETTA\_2):  
 if (A\_Numeric):  
 Result = (1 / B) \* V\_function.diff(Yy) - K\_y \* W\_Function + (1 / 2) \* (TETTA\_2 \* TETTA\_2)  
 else:  
 Result = (1 / B) \* V\_function.diff(Yy) - K\_y \* W\_Function + (1 / 2) \* (TETTA\_2 \*\* 2) - (  
 1 / (A \* B)) \* U\_function \* B.diff(Xx)  
  
 return Result  
def y\_xy(U\_function, V\_function, W\_Function,TETTA\_1,TETTA\_2):  
 if (A\_Numeric):  
 Result = (1 / A) \* V\_function.diff(Xx) + (1 / B) \* U\_function.diff(Yy) + TETTA\_1 \* TETTA\_2  
 else:  
 Result = (1 / A) \* V\_function.diff(Xx) + (1 / B) \* U\_function.diff(Yy)  
 - (1 / (A \* B)) \* V\_function \* B.diff(Xx) - (1 / (A \* B)) \* U\_function \* A.diff(Yy)  
 + TETTA\_1 \* TETTA\_2  
  
 return Result  
def e\_x\_z(ex,ksi\_1):  
 Result = ex + (-Hh/2)\*ksi\_1  
 return Result  
def e\_y\_z(ey,ksi\_2):  
 Result = ey + (-Hh/2)\*ksi\_2  
 return Result  
def y\_xy\_z(yxy,ksi\_12):  
 Result = yxy + (-Hh)\*ksi\_12  
  
 return Result  
def y\_xy\_part(U\_function, V\_function, W\_Function,TETTA\_1,TETTA\_2):  
 Result = [0] \* 3  
 Result[0] = (1 / A) \* V\_function.diff(Xx)  
 Result[1] = (1 / B) \* U\_function.diff(Yy)  
 Result[2] = TETTA\_1 \* TETTA\_2  
  
 return Result  
# Ortotrop functions  
def N\_x\_Orto(z, U\_function, V\_function, W\_Function, E\_1, nu\_12, nu\_21):  
 Result = (E\_1 / (1 - nu\_12 \* nu\_21)) \* Hh \* (  
 e\_x(U\_function, V\_function, W\_Function) + nu\_21 \* e\_y(U\_function, V\_function, W\_Function))  
 return Result  
def N\_y\_Orto(z, U\_function, V\_function, W\_Function, E\_2, nu\_12, nu\_21):  
 Result = (E\_2 / (1 - nu\_12 \* nu\_21)) \* Hh \* (  
 e\_y(U\_function, V\_function, W\_Function) + nu\_12 \* e\_x(U\_function, V\_function, W\_Function))  
 return Result  
def N\_xy\_Orto(z, U\_function, V\_function, W\_Function, G\_12):  
 Result = G\_12 \* Hh \* y\_xy(U\_function, V\_function, W\_Function)  
 return Result  
def M\_x\_Orto(W\_Function, E\_1, nu\_12, nu\_21):  
 Result = (E\_1 / (1 - nu\_12 \* nu\_21)) \* ((Hh \*\* 3) / 12) \* (  
 ksi\_1(PsiX\_function, PsiY\_function) + nu\_21 \* ksi\_2(PsiX\_function, PsiY\_function))  
 return Result  
def M\_y\_Orto(W\_Function, E\_2, nu\_12, nu\_21):  
 Result = (E\_2 / (1 - nu\_12 \* nu\_21)) \* ((Hh \*\* 3) / 12) \* (  
 ksi\_2(PsiX\_function, PsiY\_function) + nu\_12 \* ksi\_1(PsiX\_function, PsiY\_function))  
 return Result  
def M\_xy\_Orto(W\_Function, G\_12):  
 Result = 2 \* G\_12 \* ((Hh \*\* 3) / 12) \* ksi\_12(PsiX\_function, PsiY\_function)  
 return Result  
def Get\_Sigma\_x\_Orto(U\_function, V\_function, W\_Function, E\_1, nu\_12, nu\_21, z\_val, PsiX\_function, PsiY\_function):  
 Result = (E\_1 / (1 - nu\_12 \* nu\_21)) \* (  
 e\_x(U\_function, V\_function, W\_Function)  
 + nu\_21 \* e\_y(U\_function, V\_function, W\_Function)  
 + z\_val \* ksi\_1(PsiX\_function, PsiY\_function) + nu\_21 \* ksi\_2(PsiX\_function, PsiY\_function))  
 return Result  
def Get\_Sigma\_y\_Orto(U\_function, V\_function, W\_Function, E\_2, nu\_12, nu\_21, z\_val, PsiX\_function, PsiY\_function):  
 Result = (E\_2 / (1 - nu\_12 \* nu\_21)) \* (  
 e\_y(U\_function, V\_function, W\_Function)  
 + nu\_12 \* e\_x(U\_function, V\_function, W\_Function)  
 + z\_val \* ksi\_2(PsiX\_function, PsiY\_function) + nu\_12 \* ksi\_1(PsiX\_function, PsiY\_function))  
 return Result  
def Get\_Sigma\_tay\_Orto(U\_function, V\_function, W\_Function, G\_12, z\_val, PsiX\_function, PsiY\_function):  
 Result = G\_12 \* (y\_xy(U\_function, V\_function, W\_Function)  
 + 2 \* z\_val \* ksi\_12(PsiX\_function, PsiY\_function))  
  
 return Result  
# Izotrop functions  
def N\_x\_Izo(E, nu12,nu21,ex,ey):  
 result = (E / (1 - nu12 \* nu21)) \* Hh \* (ex + nu12 \* ey);  
 return result  
def N\_x\_Izo\_part(E, nu12,nu21,ex,ey):  
 Result = [0]\*2  
 Result[0] = (E / (1 - nu12 \* nu21)) \* Hh \* ex;  
 Result[1] = (E / (1 - nu12 \* nu21)) \* Hh \* nu21 \* ey;  
 return Result  
def N\_y\_Izo(E, nu12,nu21,ex,ey):  
 Result = (E / (1 - nu12 \* nu21)) \* Hh \* (ey + nu21 \* ex);  
 return Result  
def N\_y\_Izo\_part(E, nu12,nu21,ex,ey):  
 Result = [0]\*2  
 Result[0] = (E / (1 - nu12 \* nu21)) \* Hh \* ey;  
 Result[1] = (E / (1 - nu12 \* nu21)) \* Hh \* nu12 \* ex;  
 return Result  
def N\_xy\_Izo(G, nu,yxy):  
 Result = [0]\*3  
 Result[0] = G \* Hh \* yxy[0]  
 Result[1] = G \* Hh \* yxy[1]  
 Result[2] = G \* Hh \* yxy[2]  
 return Result  
def M\_x\_Izo(E, nu\_12,nu\_21,ksi1,ksi2):  
 Result = (E / (1 - nu\_12 \* nu\_21)) \* ((Hh\*\*3) / 12) \* (ksi1 + nu\_21 \* ksi2);  
 return Result  
def M\_y\_Izo(E, nu\_12,nu\_21,ksi1,ksi2):  
 Result = (E / (1 - nu\_12 \* nu\_21)) \* ((Hh\*\*3) / 12) \* (ksi2 + nu\_12 \* ksi1);  
 return Result  
def M\_xy\_Izo(G, nu, ksi12):  
 Result = 2 \* G \* ((Hh\*\*3) / 12) \* ksi12  
 return Result  
def Get\_Sigma\_x\_Izo(E, nu\_12, nu\_21, z\_val,ex,ey,KSI1,KSI2):  
 Result = (E / (1 - nu\_12 \* nu\_21)) \* (ex + nu\_21 \* ey + (-Hh/2) \* (KSI1 + nu\_21 \* KSI2))  
 return Result  
def Get\_Sigma\_y\_Izo(E, nu\_12, nu\_21, z\_val,ex,ey,KSI1,KSI2):  
 Result = (E / (1 - nu\_12 \* nu\_21)) \* (ey + nu\_12 \* ex + (-Hh/2) \* (KSI2 + nu\_12 \* KSI1))  
 return Result  
def Get\_Sigma\_tay\_Izo(G12, nu, z\_val,yxy,KSI12):  
 Result = G12 \* (yxy + 2 \* (-Hh/2) \* KSI12)  
 return Result  
def q\_function(q\_0, q\_sv):  
 Qq = Symbol('q')  
 Result = 0  
 A1 = 0  
 A\_1 = [0] \* 3  
 A\_2 = [0] \* 3  
  
 A\_1[0] = 1 - ((A\_lenght\_x + A1) / (A\_lenght\_x - A1)) \*\* 2  
 A\_1[1] = 4 \* (A\_lenght\_x + A1) / ((A\_lenght\_x - A1) \*\* 2)  
 A\_1[2] = -4 / ((A\_lenght\_x - A1) \*\* 2)  
  
 A\_2[0] = 1 - ((B\_lenght\_y + A1) / (B\_lenght\_y - A1)) \*\* 2  
 A\_2[1] = 4 \* (B\_lenght\_y + A1) / ((B\_lenght\_y - A1) \*\* 2)  
 A\_2[2] = -4 / ((B\_lenght\_y - A1) \*\* 2)  
  
 Result = Qq \* (A\_1[0] + A\_1[1] \* Xx + A\_1[2] \* (Xx \*\* 2)) \* (A\_2[0] + A\_2[1] \* Yy + A\_2[2] \* (Yy \*\* 2))  
  
 return Result  
#Rits  
def Get\_Answer\_Rits(Es\_Get,w\_coef,u\_coef,v\_coef):  
 Es = Es\_Get.copy()  
  
 Es[0] = integrate(Es[0], (Xx, Start\_integral, A\_lenght\_x))  
 Es[1] = integrate(Es[1], (Xx, Start\_integral, A\_lenght\_x))  
 Es[2] = integrate(Es[2], (Xx, Start\_integral, A\_lenght\_x))  
 Es[3] = integrate(Es[3], (Xx, Start\_integral, A\_lenght\_x))  
 Es[4] = integrate(Es[4], (Xx, Start\_integral, A\_lenght\_x))  
 Es[5] = integrate(Es[5], (Xx, Start\_integral, A\_lenght\_x))  
  
 Es[0] = (1/2) \* integrate(Es[0], (Yy, Start\_integral, B\_lenght\_y))  
 Es[1] = (1/2) \* integrate(Es[1], (Yy, Start\_integral, B\_lenght\_y))  
 Es[2] = (1/2) \* integrate(Es[2], (Yy, Start\_integral, B\_lenght\_y))  
 Es[3] = (1/2) \* integrate(Es[3], (Yy, Start\_integral, B\_lenght\_y))  
 Es[4] = (1/2) \* integrate(Es[4], (Yy, Start\_integral, B\_lenght\_y))  
 Es[5] = (1/2) \* integrate(Es[5], (Yy, Start\_integral, B\_lenght\_y))  
  
 Result = []  
 Result\_buf = []  
 Buf = []  
 Zeroes = [0] \* N \* 3  
 Buf\_Symbols = []  
  
 for index in range(1, 4):  
 Result.append(Result\_buf)  
  
 for i in range(1, N + 1):  
 Buf.append((Es[0].diff(w\_coef[i - 1]) + Es[1].diff(w\_coef[i - 1]) + Es[2].diff(w\_coef[i - 1]) + Es[3].diff(  
 w\_coef[i - 1]) + Es[4].diff(w\_coef[i - 1]) + Es[5].diff(w\_coef[i - 1])))  
 Buf\_Symbols.append(w\_coef[i - 1])  
  
 for i in range(1, N + 1):  
 Buf.append((Es[0].diff(u\_coef[i - 1]) + Es[1].diff(u\_coef[i - 1]) + Es[2].diff(u\_coef[i - 1]) + Es[3].diff(  
 w\_coef[i - 1]) + Es[4].diff(u\_coef[i - 1]) + Es[5].diff(u\_coef[i - 1])))  
 Buf\_Symbols.append(u\_coef[i - 1])  
  
 for i in range(1, N + 1):  
 Buf.append((Es[0].diff(v\_coef[i - 1]) + Es[1].diff(v\_coef[i - 1]) + Es[2].diff(v\_coef[i - 1]) + Es[3].diff(  
 v\_coef[i - 1]) + Es[4].diff(v\_coef[i - 1]) + Es[5].diff(v\_coef[i - 1])))  
 Buf\_Symbols.append(v\_coef[i - 1])  
  
 for i in range(len(Buf)):  
 Buf[i] = nsimplify(Buf[i], tolerance=1e-20).evalf(15)  
  
 for solution in linsolve(Buf, Buf\_Symbols):  
 Result = solution  
  
 return Result  
#Bub-Gal  
def get\_L1(Nx,Ny,Nxy,Qx,Tetta\_1,Tetta\_2):  
 if (A\_Numeric):  
 Result = Nx.diff(Xx) + Nxy.diff(Yy) - K\_x\*Qx +K\_x \* (Nx\*Tetta\_1 + Nxy \* Tetta\_2)  
 else:  
 Result = 0  
 return Result  
def get\_L2(Nx,Ny,Nxy,Qy,Tetta\_1,Tetta\_2):  
 if (A\_Numeric):  
 Result = Ny.diff(Yy) + Nxy.diff(Xx) - K\_y\*Qy + K\_y \* (Ny\*Tetta\_2 + Nxy \* Tetta\_1)  
 else:  
 Result = 0  
 return Result  
def get\_L3(Nx,Ny,Nxy,Tetta\_1,Tetta\_2,Qx,Qy,q):  
 if (A\_Numeric):  
 Result = K\_x\*Nx + K\_y\*Ny - (Nx\*Tetta\_1 + Nxy\*Tetta\_2).diff(Xx) \  
 - (Ny\*Tetta\_2 + Nxy\*Tetta\_1).diff(Yy) + Qx.diff(Xx) + Qy.diff(Yy) + q  
 else:  
 Result = 0  
 return Result  
def get\_L4(Mx,Mxy,Qx):  
 if (A\_Numeric):  
 Result = Mx.diff(Xx) + Mxy.diff(Yy) - Qx  
  
 else:  
 Result = 0  
 return Result  
def get\_L5(My,Mxy,Qy):  
 if (A\_Numeric):  
 Result = My.diff(Yy) + Mxy.diff(Xx) - Qy  
 else:  
 Result = 0  
 return Result  
def Get\_Es\_Bub(Nx,Ny,Nxy,Mx,My,Mxy,Qx,Qy,Tetta\_1,Tetta\_2,Q\_find):  
 Es = [0]\*N\*5  
  
 L1 = get\_L1(Nx,Ny,Nxy,Qx,Tetta\_1,Tetta\_2)  
 L2 = get\_L2(Nx,Ny,Nxy,Qy,Tetta\_1,Tetta\_2)  
 L3 = get\_L3(Nx, Ny, Nxy, Tetta\_1, Tetta\_2, Qx, Qy, 0)  
 L4 = get\_L4(Mx,Mxy,Qx)  
 L5 = get\_L5(My,Mxy,Qy)  
  
 for j in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 #print("Now = ",(j - 1) \* N\_x + i - 1)  
 Es[(j - 1) \* N\_x + i - 1] = L3 \* Get\_w\_sin\_x(i) \* Get\_w\_sin\_y(j)  
 Es[(j - 1) \* N\_x + i - 1 + N] = L2 \* Get\_w\_sin\_x(i) \* Get\_w\_sin\_y\_2(j) #V  
 Es[(j - 1) \* N\_x + i - 1 + N \* 2] = L1 \* Get\_w\_sin\_x\_2(i) \* Get\_w\_sin\_y(j) #U  
 Es[(j - 1) \* N\_x + i - 1 + N \* 3] = L4 \* Get\_w\_cos\_x(i) \* Get\_w\_sin\_y(j) #PsiX  
 Es[(j - 1) \* N\_x + i - 1 + N \* 4] = L5 \* Get\_w\_sin\_x(i) \* Get\_w\_cos\_y(j) #PsiY  
  
 #print("ES",Es)  
 return Es  
def Get\_Jacobian\_Bubnov(Function\_E, Result\_w):  
 #print("Jacodi start)")  
 Jacobian = [0] \* N \* 5  
 Def\_Function = Function\_E  
 Symbol\_Function = [0] \* N \* 5  
  
 #print("Function E",Function\_E)  
 for i in range(0, N):  
 Symbol\_Function[i] = 'w' + str(i + 1)  
 Symbol\_Function[i + N] = 'u' + str(i + 1)  
 Symbol\_Function[i + N \* 2] = 'v' + str(i + 1)  
 Symbol\_Function[i + N \* 3] = 'PsiX' + str(i + 1)  
 Symbol\_Function[i + N \* 4] = 'PsiY' + str(i + 1)  
  
 for i in range(0, N \* 5):  
 Jacobian[i] = [0] \* N \* 5  
 Def\_Function[i] = Function\_E[i]  
 #print("D[",i,"] =",Def\_Function[i])  
  
 for row in range(0, N \* 5):  
 for column in range(0, N \* 5):  
 Jacobian[row][column] = Def\_Function[row].diff(Symbol\_Function[column])  
 for W\_coefs in range(0, N \* 5):  
 Jacobian[row][column] = Jacobian[row][column].subs((Symbol\_Function[W\_coefs]), (Result\_w[W\_coefs]))  
 #print("J[", row, "][", column, "] =", Jacobian[row][column])  
 #print("Result J =", Jacobian)  
 return Jacobian  
def Get\_New\_iterarion\_Bubnov(Function\_E, Jackobi\_inv, Last\_step\_X, a):  
 Def\_Function = [0] \* N \* 5  
 Symbol\_Function = [0] \* N \* 5  
 #print( "A = ", a)  
 for i in range(0, N):  
 Symbol\_Function[i] = 'w' + str(i + 1)  
 Symbol\_Function[i + N] = 'u' + str(i + 1)  
 Symbol\_Function[i + N \* 2] = 'v' + str(i + 1)  
 Symbol\_Function[i + N \* 3] = 'PsiX' + str(i + 1)  
 Symbol\_Function[i + N \* 4] = 'PsiY' + str(i + 1)  
  
 for i in range(0, N \* 5):  
 Def\_Function[i] = Function\_E[i]  
 #print(i, " Func e =",Function\_E[i])  
 #for i in range(0, N \* 5):  
 #for j in range(0, N \* 5):  
 #Def\_Function[i] = Function\_E[i].diff(Symbol\_Function[j])  
 #print(Jackobi\_inv[i][j])  
 for i in range(0, N \* 5):  
 for W\_coefs in range(0, N \* 5):  
 Def\_Function[i] = Def\_Function[i].subs(Symbol\_Function[W\_coefs], (Last\_step\_X[W\_coefs]))  
 #print("Def\_Function[", i, "] = ", Def\_Function[i])  
  
 Def\_Function = sym.Matrix(Def\_Function)  
 W = sym.Matrix(Last\_step\_X)  
 #print(Last\_step\_X)  
 #print("w = ", W)  
 # print("W =" ,W)  
 # print("a =",a)  
 # print("Ja =",Jackobi\_inv)  
 # print("DF =",Def\_Function)  
 Result = W - a \* (Jackobi\_inv \* Def\_Function)  
 #W = W \* Jackobi\_inv  
 #print("Result = ", Result)  
 return Result  
def Bubnov\_Iter(Function\_E, eps, w0):  
 All\_Results = []  
 Res\_now = w0  
 Now\_eps = 1  
 Check\_Loop = 1  
 Count\_Iterions = 0  
 New\_Eps = [0] \* N\*5  
  
 while (Now\_eps > eps):  
 Count\_Iterions += 1  
 Max\_eps = 0  
  
 Jacobi = Get\_Jacobian\_Bubnov(Function\_E, Res\_now)  
 print("Jacobi = ", Jacobi[0])  
 Jackobi\_Matrix = sym.Matrix(Jacobi)  
 #print("j =",Jackobi\_Matrix)  
 #for i in range(5):  
 #for j in range(5):  
 #print("Jacobi = ", Jackobi\_Matrix[i][j])  
  
 Jacobi\_Invariant = Jackobi\_Matrix.inv()  
 print("Jacobi = ", Jacobi\_Invariant)  
  
  
 Res\_new = Get\_New\_iterarion\_Bubnov(Function\_E, Jacobi\_Invariant, Res\_now, Check\_Loop)  
 print("Res\_now = ",Res\_new)  
 np.array(Res\_new).astype(np.float64)  
  
 for i in range(0, N\*5):  
 New\_Eps[i] = abs(Res\_new[i] - Res\_now[i])  
 if (New\_Eps[i] > Max\_eps):  
 Max\_eps = New\_Eps[i]  
  
 #print("max = ",Max\_eps)  
 Res\_Last\_now = Res\_now  
 Now\_eps = Max\_eps  
 Res\_now = Res\_new  
 All\_Results.append(Res\_now)  
  
 if Count\_Iterions > 10:  
 Check\_Loop = Check\_Loop / 10  
 Res\_now = Res\_Last\_now  
 if All\_Results[Count\_Iterions - 3][0] < All\_Results[Count\_Iterions - 2][0] < \  
 All\_Results[Count\_Iterions - 1][0]:  
 if Check\_Loop != 1:  
 Check\_Loop \*= 10  
 Res\_now = Res\_new  
 if All\_Results[Count\_Iterions - 3][0] > All\_Results[Count\_Iterions - 2][0] > \  
 All\_Results[Count\_Iterions - 1][0]:  
 if Check\_Loop != 1:  
 Check\_Loop \*= 10  
 Res\_now = Res\_new  
 if Count\_Iterions > 50:  
 Now\_eps = 0  
 #print("New\_iteration = ", Res\_now)  
 #print("R\_now",Res\_now)  
  
 return Res\_now  
def Bubnov\_Loop(Nx,Ny,Nxy,Mx,My,Mxy,Qx,Qy,Tetta\_1,Tetta\_2,Q\_find):  
 print("Start loop")  
 # Q\_now = 0.01  
 # Q\_now = 0  
 q\_T = 3.14 / 10  
 Q\_step = q\_T / 2  
 Miz = 0  
 Q\_now = 0.01  
 q\_for\_graph = []  
 w\_for\_graph = []  
 q\_for\_graph.append(0)  
 w\_for\_graph.append(0)  
 W\_Result = [0] \* (1000)  
 W\_Result[0] = [0.0001] \* N \* 5  
  
 Es = [0] \* N \* 5  
 Es = Get\_Es\_Bub(Nx,Ny,Nxy,Mx,My,Mxy,Qx,Qy,Tetta\_1,Tetta\_2,Q\_find)  
  
 queue = multiprocessing.Queue()  
 timer = time.time()  
 p = [0] \* (N \* 5-N+1)  
  
 Simp = 1  
 if Simp == 1:  
 for i in range(0, N \* 5):  
 print("Es ",i,"= ",Es[i])  
 p[i] = multiprocessing.Process(target=intagrete\_Es\_Simp\_Bubnov, args=(Es[i], queue, i,))  
 if Simp == 0:  
 for i in range(0, N \* 5):  
 p[i] = multiprocessing.Process(target=intagrete\_Es, args=(Es[i], queue, i,))  
 for i in range(0, N \* 5):  
 p[i].start()  
  
 t = 0  
 if N == 1:  
 t = 100  
 if N == 4:  
 t = 360  
  
 p[0].join(timeout=t)  
 print("p", 0, " - join")  
 for i in range(1, N \* 5):  
 p[i].join(timeout=t)  
 print("p", i, " - join")  
  
 j = N  
 Buf\_Function = [0] \* N \* 5  
 while queue.qsize() > 0:  
 buf = queue.get()  
 print(buf)  
 function = buf[0]  
 index = buf[1]  
 Buf\_Function[index] = (1/2) \* function  
 print("j =", j, " = ", buf)  
 j += 1  
 print('Time %.6f' % (time.time() - timer))  
  
 j = 1  
 while (Q\_now < 6):  
 Dop\_L3 = Q\_now  
 Dop\_Integrals = [0]\*N  
 for jj in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 Dop\_Integrals[(jj - 1) \* N\_x + i - 1] = Dop\_L3 \* Get\_w\_sin\_x(i) \* Get\_w\_sin\_y(jj) # W  
  
 print("Q\_now = ", Q\_now)  
 print("Start integrate 7")  
 for i in range(0,N):  
 Dop\_Integrals[i] = integrate(Dop\_Integrals[i], (Xx, Start\_integral, A\_lenght\_x))  
 Buf\_Function[i] += integrate(Dop\_Integrals[i], (Yy, Start\_integral, B\_lenght\_y))  
 print("End 7 ")  
  
 W\_Result[j] = Bubnov\_Iter(Buf\_Function, eps, W\_Result[j-1])  
  
 W\_val = [] \* N  
 U\_val = [] \* N  
 V\_val = [] \* N  
 PsiX\_val = [] \* N  
 PsiY\_val = [] \* N  
  
 for i in range(1, N + 1):  
 W\_val.append(W\_Result[j][i - 1])  
 U\_val.append(W\_Result[j][i - 1 + N])  
 V\_val.append(W\_Result[j][i - 1 + 2 \* N])  
 PsiX\_val.append(W\_Result[j][i - 1 + 3 \* N])  
 PsiY\_val.append(W\_Result[j][i - 1 + 4 \* N])  
  
 for i in range(1, N + 1):  
 print("W(",j,")(",i,") = ",W\_val[i - 1])  
 for i in range(1, N + 1):  
 print("U(", j, ")(", i, ") = ", U\_val[i - 1])  
 for i in range(1, N + 1):  
 print("V(", j, ")(", i, ") = ", V\_val[i - 1])  
 for i in range(1, N + 1):  
 print("PsiX(", j, ")(", i, ") = ", PsiX\_val[i - 1])  
 for i in range(1, N + 1):  
 print("PsiY(", j, ")(", i, ") = ", PsiY\_val[i - 1])  
  
 #Miz = Draw\_3d\_Sigmas\_main('Sigma\_i', W\_val, 1, U\_function, V\_function, W\_Function, PsiX\_function, PsiY\_function,  
 #z,W\_val, U\_val, V\_val, PsiX\_val, PsiY\_val,Sigma\_x,Sigma\_y,Sigma\_tay)  
  
 New\_w = Get\_W\_Plane(A\_lenght\_x / 2, B\_lenght\_y / 2, 0, W\_val, 3)  
 Last\_w = w\_for\_graph[len(w\_for\_graph)-1]  
  
 #New\_w = Get\_W\_Plane(A\_lenght\_x / 2, B\_lenght\_y / 2, 0, W\_val, 3)  
 #print("W(l/2) = ", New\_w)  
 #New\_w = Get\_W\_Plane(A\_lenght\_x / 4, B\_lenght\_y / 4, 0, W\_val, 3)  
 #print("W(l/4) = ", New\_w)  
  
 Q\_now += Q\_step  
 j += 1  
 print("Miz = ", Miz)  
  
 if New\_w > Last\_w:  
 w\_for\_graph.append(Get\_W\_Plane(A\_lenght\_x / 2, B\_lenght\_y / 2, 0, W\_val, 3))  
 q\_for\_graph.append(3 \* (Q\_now - Q\_step))  
 else:  
 print("trouble")  
 continue  
  
 Ww = [0] \* 2  
 Qq = [0] \* 2  
 Ww.append(w\_for\_graph[0])  
 Ww.append(w\_for\_graph[len(w\_for\_graph)-1])  
 Qq.append(q\_for\_graph[0])  
 Qq.append(q\_for\_graph[len(q\_for\_graph) - 1])  
 print("Plot =")  
 plt.plot(w\_for\_graph, q\_for\_graph)  
 #plt.plot(Ww, Qq)  
 plt.show()  
 print("Get\_Bub")  
def Bubnov\_byPoint(Nx,Ny,Nxy,Mx,My,Mxy,Qx,Qy,Tetta\_1,Tetta\_2,Q\_find):  
 print("Start loop")  
 # Q\_now = 0.01  
 # Q\_now = 0  
 q\_T = 3.14 / 10  
 Q\_step = q\_T / 2  
 Miz = 0  
 Q\_now = 0.01  
 q\_for\_graph = []  
 w\_for\_graph = []  
 q\_for\_graph.append(0)  
 w\_for\_graph.append(0)  
 W\_Result = [0] \* (1000)  
 W\_Result[0] = [0.0001] \* N \* 5  
  
 Es = [0] \* N \* 5  
 Es = Get\_Es\_Bub(Nx,Ny,Nxy,Mx,My,Mxy,Qx,Qy,Tetta\_1,Tetta\_2,Q\_find)  
  
 queue = multiprocessing.Queue()  
 timer = time.time()  
 p = [0] \* (N \* 5)  
 Buf\_Function = [0] \* N \* 5  
  
 for index\_thread in range (N):  
 Simp = 1  
 if Simp == 1:  
 for i in range(index\_thread \* 5,index\_thread \* 5 + 5):  
 print("Es ",i,"= ",Es[i])  
 p[i] = multiprocessing.Process(target=intagrete\_Es\_Simp\_Bubnov, args=(Es[i], queue, i,))  
 if Simp == 0:  
 for i in range(index\_thread \* 5,index\_thread \* 5 + 5):  
 p[i] = multiprocessing.Process(target=intagrete\_Es, args=(Es[i], queue, i,))  
 for i in range(index\_thread \* 5,index\_thread \* 5 + 5):  
 print("p ",i," start")  
 p[i].start()  
  
 t = 0  
 if index\_thread == 0:  
 if N == 1:  
 t = 100  
 if N == 4:  
 t = 550  
 else:  
 if N == 1:  
 t = 100  
 if N == 4:  
 t = 260  
  
 p[index\_thread \* 5].join()  
 print("p", 0, " - join")  
 for i in range(index\_thread \* 5+1,index\_thread \* 5 + 5):  
 p[i].join()  
 print("p", i, " - join")  
  
 j = N  
 while queue.qsize() > 0:  
 buf = queue.get()  
 function = buf[0]  
 index = buf[1]  
 Buf\_Function[index] = function  
 print("j =", j, " = ", buf)  
 j += 1  
 print('Time %.6f' % (time.time() - timer))  
  
 j = 1  
 #Dop\_L3 = Q\_find  
 Dop\_L3 = 3.14  
 Dop\_Integrals = [0]\*N  
 for jj in range(1, N\_x + 1):  
 for i in range(1, N\_y + 1):  
 Dop\_Integrals[(jj - 1) \* N\_x + i - 1] = Dop\_L3 \* Get\_w\_sin\_x(i) \* Get\_w\_sin\_y(jj) # W  
  
 print("Q\_now = ", Q\_now)  
 print("Start integrate 7")  
 for i in range(0,N):  
 Dop\_Integrals[i] = integrate(Dop\_Integrals[i], (Xx, Start\_integral, A\_lenght\_x))  
 Buf\_Function[i] += integrate(Dop\_Integrals[i], (Yy, Start\_integral, B\_lenght\_y))  
 print("End 7 ")  
  
 W\_Result[j] = Bubnov\_Iter(Buf\_Function, eps, W\_Result[j-1])  
  
 W\_val = [] \* N  
 U\_val = [] \* N  
 V\_val = [] \* N  
 PsiX\_val = [] \* N  
 PsiY\_val = [] \* N  
  
 for i in range(1, N + 1):  
 W\_val.append(W\_Result[j][i - 1])  
 U\_val.append(W\_Result[j][i - 1 + N])  
 V\_val.append(W\_Result[j][i - 1 + 2 \* N])  
 PsiX\_val.append(W\_Result[j][i - 1 + 3 \* N])  
 PsiY\_val.append(W\_Result[j][i - 1 + 4 \* N])  
  
 for i in range(1, N + 1):  
 print("W(",j,")(",i,") = ",W\_val[i - 1])  
 for i in range(1, N + 1):  
 print("U(", j, ")(", i, ") = ", U\_val[i - 1])  
 for i in range(1, N + 1):  
 print("V(", j, ")(", i, ") = ", V\_val[i - 1])  
 for i in range(1, N + 1):  
 print("PsiX(", j, ")(", i, ") = ", PsiX\_val[i - 1])  
 for i in range(1, N + 1):  
 print("PsiY(", j, ")(", i, ") = ", PsiY\_val[i - 1])  
  
 #Miz = Draw\_3d\_Sigmas\_main('Sigma\_i', W\_val, 1, U\_function, V\_function, W\_Function, PsiX\_function, PsiY\_function,  
 #z,W\_val, U\_val, V\_val, PsiX\_val, PsiY\_val,Sigma\_x,Sigma\_y,Sigma\_tay)  
  
 Res\_w = Get\_W\_Plane(A\_lenght\_x / 2, B\_lenght\_y / 2, 0, W\_val, 3)  
 print("W(l/2) =", Res\_w)  
 Res\_w = Get\_W\_Plane(A\_lenght\_x / 4, B\_lenght\_y / 4, 0, W\_val, 3)  
 print("W(l/4) =", Res\_w)  
  
 #New\_w = Get\_W\_Plane(A\_lenght\_x / 2, B\_lenght\_y / 2, 0, W\_val, 3)  
 #print("W(l/2) = ", New\_w)  
 #New\_w = Get\_W\_Plane(A\_lenght\_x / 4, B\_lenght\_y / 4, 0, W\_val, 3)  
 #print("W(l/4) = ", New\_w)  
  
 j += 1  
 print("Miz = ", Miz)  
  
 print("Get\_Bub")  
#Nuton  
def Get\_Jacobian\_Nuton(Function\_E):  
 Jacobian = []  
 Def\_Function = [0] \* N \* 5  
 Symbol\_Function = [0] \* N \* 5  
  
 timer = time.time()  
 for i in range(0, N):  
 Symbol\_Function[i] = 'w' + str(i + 1)  
 Symbol\_Function[i + N] = 'u' + str(i + 1)  
 Symbol\_Function[i + N \* 2] = 'v' + str(i + 1)  
 Symbol\_Function[i + N \* 3] = 'PsiX' + str(i + 1)  
 Symbol\_Function[i + N \* 4] = 'PsiY' + str(i + 1)  
 if Multi == 1:  
 Data = []  
 for i in range(0, N \* 5):  
 Data.append((Function\_E,Symbol\_Function[i], i))  
 with mp.Pool(N \* 5) as p:  
 p\_result = p.map(get\_diff\_2, Data)  
 for i in range(0, N \* 5):  
 Def\_Function[i] = p\_result[i]  
 #print('First\_Jacobi\_multi = Time %.6f' % (time.time() - timer))  
 else:  
 for i in range(0, N \* 5):  
 Def\_Function[i] = Function\_E.diff(Symbol\_Function[i])  
 #print('First\_Jacobi\_not multi = Time %.6f' % (time.time() - timer))  
  
 timer = time.time()  
 Data = []  
 if Multi == 1:  
 for row in range(0, N \* 5):  
 Jacobian.append([])  
 for column in range(0, N \* 5):  
 Jacobian[row].append([])  
 Data.append((Def\_Function[row], Symbol\_Function[column], row, column))  
 with mp.Pool(60) as p:  
 p\_result = p.map(get\_diff\_2arg\_2, Data)  
 for row in range(0, N \* 5):  
 for column in range(0, N \* 5):  
 Jacobian[row][column] = p\_result[row \* (N \* 5) + column][0]  
 #print('Second\_Jacobi\_multi = Time %.6f' % (time.time() - timer))  
 else:  
 for row in range(0, N \* 5):  
 Jacobian.append([])  
 for column in range(0, N \* 5):  
 Jacobian[row].append([])  
 Jacobian[row][column] = Def\_Function[row].diff(Symbol\_Function[column])  
 #print('Second\_Jacobi\_not\_multi = Time %.6f' % (time.time() - timer))  
  
 return Def\_Function,Jacobian  
def Get\_New\_iterarion\_Nuton(Def\_function\_get, Jackobi\_inv, Last\_step\_X, a):  
 Def\_Function = Def\_function\_get.copy()  
 Symbol\_Function = [0] \* N \* 5  
 #print( "A = ", a)  
 for i in range(0, N):  
 Symbol\_Function[i] = 'w' + str(i + 1)  
 Symbol\_Function[i + N] = 'u' + str(i + 1)  
 Symbol\_Function[i + N \* 2] = 'v' + str(i + 1)  
 Symbol\_Function[i + N \* 3] = 'PsiX' + str(i + 1)  
 Symbol\_Function[i + N \* 4] = 'PsiY' + str(i + 1)  
  
 Result\_function = []  
 Symbol\_data = []  
 Data = []  
 for W\_coefs in range(0, N \* 5):  
 Symbol\_data.append((Symbol\_Function[W\_coefs], Last\_step\_X[W\_coefs]))  
 if Multi ==1 and N!=1:  
 for i in range(0, N \* 5):  
 for W\_coefs in range(0, N \* 5):  
 Def\_Function[i] = Def\_Function[i].subs(Symbol\_Function[W\_coefs], (Last\_step\_X[W\_coefs]))  
 for i in range(0, N \* 5):  
 Data.append((Def\_Function[i], Symbol\_data))  
 with mp.Pool(60) as p:  
 p\_result = p.map(Sub\_data, Data)  
 for i in range(0, N \* 5):  
 Result\_function.append(p\_result[i])  
 else:  
 for i in range(0, N \* 5):  
 Result\_function.append(0)  
 for W\_coefs in range(0, N \* 5):  
 Result\_function[i] = Def\_Function[i].subs(Symbol\_data)  
  
 Result\_function = sym.Matrix(Result\_function)  
 W = sym.Matrix(Last\_step\_X)  
  
 step = a  
 buf = Jackobi\_inv \* Result\_function  
 check =1  
 new\_step =0  
 while(check):  
 check = 0  
 if abs((buf[0]\*step)/4) > abs(W[0]) or (W[0] - step \* buf[0]) < 0:  
 step /= 2  
 check = 1  
 new\_step = 1  
  
  
 Result = W - step \* buf  
 print("A = ", step)  
 if new\_step == 1:  
 step\*=2  
 step+=step/10  
 return Result,step  
def Nuton\_Loop(Es\_Get,W\_Function\_get,sigma\_x,sigma\_y,sigma\_tay):  
 print("Start loop")  
 Q\_now = Q\_step  
  
 Miz = []  
 color = ['-.b','-b','-.r','-r','-.m','-m']  
 q\_for\_graph = []  
 w\_for\_graph = []  
 q\_for\_graph\_4 = []  
 w\_for\_graph\_4 = []  
 for i in range(0,3):  
 Miz.append([])  
 Miz[i].append([])  
 Miz[i].append(0)  
 Miz[i].append(0)  
 Miz[i].append(0)  
 Miz[i].append(0)  
 w\_for\_graph.append([])  
 q\_for\_graph.append([])  
 q\_for\_graph\_4.append([])  
 w\_for\_graph\_4.append([])  
 q\_for\_graph[i].append(0)  
 w\_for\_graph[i].append(0)  
 q\_for\_graph\_4[i].append(0)  
 w\_for\_graph\_4[i].append(0)  
  
 W\_Result = [0] \* (1000)  
 W\_Result[0] = [0.001] \* N\*5  
  
 Q\_function = q\_function(Q\_now,0)  
 Es\_Get[7] = (-2) \* (Q\_function \* W\_Function\_get)  
  
 timer = time.time()  
 Es = Es\_Get.copy()  
 p\_result = []  
 Data = []  
 Buf\_Function = 0  
 for i in range(0, Count\_Es):  
 Data.append((Es[i], i))  
 with mp.Pool(Count\_Es) as p:  
 p\_result = p.map(intagrete\_Es\_Simp\_2, Data)  
 for i in range(0, Count\_Es):  
 Buf\_Function += (1 / 2) \* p\_result[i]  
 print('End integrade = Time %.6f' % (time.time() - timer))  
  
 for H\_value in range(1,4):  
 j=1  
 index\_g = 0  
 Q\_now = Q\_step  
 Miz\_stop = 0  
 Miz\_now = 0  
 new\_sigma\_x = sigma\_x.subs([(Hh, h \* H\_value)])  
 new\_sigma\_y = sigma\_y.subs([(Hh, h \* H\_value)])  
 new\_sigma\_tay = sigma\_tay.subs([(Hh, h \* H\_value)])  
 # Цикл по разным q  
 while (Q\_now < q\_find\*H\_value):  
 print("\nIteration =", j, " and h =",H\_value)  
 print("Q\_now = ", Q\_now,"\n")  
  
 New\_Buf\_Function = Buf\_Function.subs([('q',Q\_now),(Hh,h\*H\_value)])  
  
 #print("W\_Result\_send = ",W\_Result[j-1])  
 W\_Result[j] = Nuton\_Iter(New\_Buf\_Function, eps, W\_Result[j-1])  
  
 W\_val = [] \* N  
 U\_val = [] \* N  
 V\_val = [] \* N  
 PsiX\_val = [] \* N  
 PsiY\_val = [] \* N  
  
 for i in range(1, N + 1):  
 W\_val.append(W\_Result[j][i - 1])  
 U\_val.append(W\_Result[j][i - 1 + N])  
 V\_val.append(W\_Result[j][i - 1 + 2 \* N])  
 PsiX\_val.append(W\_Result[j][i - 1 + 3 \* N])  
 PsiY\_val.append(W\_Result[j][i - 1 + 4 \* N])  
 for i in range(1, N + 1):  
 print("W(",j,")(",i,") = ",W\_val[i - 1])  
 for i in range(1, N + 1):  
 print("U(", j, ")(", i, ") = ", U\_val[i - 1])  
 for i in range(1, N + 1):  
 print("V(", j, ")(", i, ") = ", V\_val[i - 1])  
 for i in range(1, N + 1):  
 print("PsiX(", j, ")(", i, ") = ", PsiX\_val[i - 1])  
 for i in range(1, N + 1):  
 print("PsiY(", j, ")(", i, ") = ", PsiY\_val[i - 1])  
  
 Q\_now += Q\_step\*H\_value  
 j += 1  
 w\_2 = Get\_W\_Plane(A\_lenght\_x / 2, B\_lenght\_y / 2, 0, W\_val, 3)  
 w\_4 = Get\_W\_Plane(A\_lenght\_x / 4, B\_lenght\_y / 4, 0, W\_val, 3)  
  
 #Miz\_now,x\_get,y\_get = get\_Mizz(W\_val, U\_val, V\_val, PsiX\_val, PsiY\_val,new\_sigma\_x,new\_sigma\_y,new\_sigma\_tay)  
 Miz\_now, x\_get, y\_get = get\_Mizz\_2(W\_val, U\_val, V\_val, PsiX\_val, PsiY\_val, new\_sigma\_x, new\_sigma\_y, new\_sigma\_tay)  
 #print("Mizz =",Miz\_now)  
 #print("x\_get =", x\_get)  
 #print("y\_get =", y\_get)  
  
 if w\_2 < w\_for\_graph[H\_value-1][index\_g]:  
 Q\_now -=(Q\_step\*H\_value)/2  
 else:  
 index\_g+=1  
 w\_for\_graph[H\_value-1].append(w\_2)  
 q\_for\_graph[H\_value-1].append(Q\_now - Q\_step\*H\_value)  
 w\_for\_graph\_4[H\_value-1].append(w\_4)  
 q\_for\_graph\_4[H\_value-1].append(Q\_now - Q\_step\*H\_value)  
 print(" M =",Miz\_now," and stop =",Miz\_stop)  
 if Miz\_now > 1 and Miz\_stop == 0 and j>3:  
 Miz[H\_value - 1][0] = Q\_now - Q\_step\*H\_value  
 Miz[H\_value - 1][1] = Miz\_now  
 Miz[H\_value - 1][2] = x\_get  
 Miz[H\_value - 1][3] = y\_get  
 print("STOP!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")  
 Miz\_stop = 1  
 if Miz\_stop !=1:  
 Miz[H\_value - 1][0] = Q\_now - Q\_step \* H\_value  
 Miz[H\_value - 1][1] = Miz\_now  
  
 plt.plot(w\_for\_graph[H\_value-1], q\_for\_graph[H\_value-1],color[H\_value\*2-1])  
 plt.plot(w\_for\_graph\_4[H\_value-1], q\_for\_graph\_4[H\_value-1],color[(H\_value-1)\*2])  
 #Ww = [0] \* 2  
 #Qq = [0] \* 2  
 #Ww.append(w\_for\_graph[0])  
 #Ww.append(w\_for\_graph[len(w\_for\_graph)-1])  
 #Qq.append(q\_for\_graph[0])  
 #Qq.append(q\_for\_graph[len(q\_for\_graph) - 1])  
 for i in range(0,3):  
 print("When H =",h\*(i + 1 )," Miz = ",Miz[i][1]," when q =",Miz[i][0]," in point x,y =(",Miz[i][2],",",Miz[i][3],") !")  
 if Change == 1:  
 plt.legend(['T300/976 a in l/2', 'T300/976 a in l/4', 'T300/976 2\*a in l/2', 'T300/976 a in l/4',  
 'T300/976 3\*a in l/2', 'T300/976 3\*a in l/4'])  
 if Change == 2:  
 plt.legend(['Org\_glass a in l/2', 'Org\_glass a in l/4', 'Org\_glass 2\*a in l/2', 'Org\_glass a in l/4',  
 'Org\_glass 3\*a in l/2', 'Org\_glass 3\*a in l/4'])  
 if Change == 3:  
 plt.legend(['Still a in l/2', 'Still a in l/4', 'Still 2\*a in l/2', 'Still a in l/4', 'Still 3\*a in l/2',  
 'Still 3\*a in l/4'])  
  
 plt.title("График зависимости W-q")  
 plt.xlabel("W [мм]")  
 plt.ylabel("q [кПа]")  
 plt.show()  
 return W\_Result  
def Nuton\_byPoint(Es\_Get,W\_Function\_get):  
 print("Start loop")  
 print("Q\_find = ", q\_find)  
 Q\_function = q\_function(q\_find,0)  
 Q\_function = Q\_function.subs('q',q\_find)  
 Es\_Get[7] = (-2) \* (Q\_function \* W\_Function\_get)  
 Es = Es\_Get.copy()  
  
 W\_Result = [0] \* (5)  
 W\_Result[0] = [0.0001] \* N\*5  
 #W\_Result[0][0] = 3  
 #print(W\_Result[0])  
 #W\_Result[0] = [3.74885474336874, -0.0279630259665207, 0.0541634218784015, -0.0907776564867707, -0.0303405809568086, -0.00120203304247375, 0.00502268166794714, 0.00107163810758357, -0.0387194608081201, 0.00691748440476835, -0.00412977762537998, 0.00163838010816346, -0.0588816111494670, 0.00131429124355399, -0.000851701731114988, 0.00427798479980411, -0.0588769537235213, 0.000437378198980303, -0.00255836617573380, 0.00427844847406291]  
 W\_Result[0] = [0.017786911642670, -0.000564657955444373, -0.00398439759420913, -0.00120268363699261,  
 0.000895996863933102, 0.000965897325377122, 0.000969422666680191, 0.000981538950014033,  
 0.000846484633837755, 0.000965115282362594, 0.000976416650144447, 0.000983355572424134,  
 0.000697066265024303, 0.00104930782231455, 0.00105461033201082, 0.00107937658489713,  
 0.000697082825784533, 0.00100089631628238, 0.00121043813865665, 0.00107937507012975]  
 #W\_Result[0] = [0.00453194824996379, 0.000686909891288986, 3.30388525480721e-5, 0.000558229220868899,  
 # 0.000976155064805989, 0.000990027925391850, 0.000990664864102216, 0.000993139076428632,  
 # 0.000966434033902347, 0.000989863554816704, 0.000991997402871729, 0.000993515508917947,  
 # 0.000936588041744951, 0.00100657122559083, 0.00100725325523264, 0.00101263564456488,  
 # 0.000936591321778728, 0.000996982873707688, 0.00103738006690587, 0.00101263534450048]  
 #print(W\_Result[0])  
  
 timer = time.time()  
 p\_result =[]  
 Data = []  
 Buf\_Function = 0  
 for i in range(0,Count\_Es):  
 Data.append((Es[i],i))  
 with mp.Pool(Count\_Es) as p:  
 p\_result = p.map(intagrete\_Es\_Simp\_2,Data)  
 for i in range(0,Count\_Es):  
 Buf\_Function += (1 / 2) \* p\_result[i]  
 print('End integrade = Time %.6f' % (time.time() - timer))  
  
 Buf\_Function = Buf\_Function.subs(Hh,h)  
 W\_Result[1] = Nuton\_Iter(Buf\_Function, eps, W\_Result[0])  
  
 W\_val = [] \* N  
 U\_val = [] \* N  
 V\_val = [] \* N  
 PsiX\_val = [] \* N  
 PsiY\_val = [] \* N  
  
 for i in range(1, N + 1):  
 W\_val.append(W\_Result[1][i - 1])  
 U\_val.append(W\_Result[1][i - 1 + N])  
 V\_val.append(W\_Result[1][i - 1 + 2 \* N])  
 PsiX\_val.append(W\_Result[1][i - 1 + 3 \* N])  
 PsiY\_val.append(W\_Result[1][i - 1 + 4 \* N])  
 j=0  
 for i in range(1, N + 1):  
 print("W(",1,")(",i,") = ",W\_val[i - 1])  
 for i in range(1, N + 1):  
 print("U(", 1, ")(", i, ") = ", U\_val[i - 1])  
 for i in range(1, N + 1):  
 print("V(", 1, ")(", i, ") = ", V\_val[i - 1])  
 for i in range(1, N + 1):  
 print("PsiX(", 1, ")(", i, ") = ", PsiX\_val[i - 1])  
 for i in range(1, N + 1):  
 print("PsiY(", 1, ")(", i, ") = ", PsiY\_val[i - 1])  
  
 Res\_w\_2 = Get\_W\_Plane(A\_lenght\_x / 2, B\_lenght\_y / 2, 0, W\_val, 3)  
 Res\_w\_4 = Get\_W\_Plane(A\_lenght\_x / 4, B\_lenght\_y / 4, 0, W\_val, 3)  
 print("W(l/2) =", Res\_w\_2)  
 print("W(l/4) =", Res\_w\_4)  
 return W\_Result[1]  
# Nut iter  
def Nuton\_Iter(Function\_E, eps, w0):  
 All\_Results = []  
 Result\_now = w0  
  
 Now\_eps = 1  
 Step\_x = 1  
 Count\_Iterions = 0  
  
 New\_Eps = [0] \* N\*5  
 Eps\_all = 1  
  
 #Create symbol massiv  
 Symbol\_Function = [0] \* N \* 5  
 for i in range(0, N):  
 Symbol\_Function[i] = 'w' + str(i + 1)  
 Symbol\_Function[i + N] = 'u' + str(i + 1)  
 Symbol\_Function[i + N \* 2] = 'v' + str(i + 1)  
 Symbol\_Function[i + N \* 3] = 'PsiX' + str(i + 1)  
 Symbol\_Function[i + N \* 4] = 'PsiY' + str(i + 1)  
 Symbol\_data = []  
 for W\_coefs in range(0, N \* 5):  
 Symbol\_data.append((Symbol\_Function[W\_coefs], Result\_now[W\_coefs]))  
  
 #Get DF and Jacobi  
 timer = time.time()  
 Def\_Function,Jacobi = Get\_Jacobian\_Nuton(Function\_E)  
  
 #print('Get jacobi all time = Time %.6f' % (time.time() - timer))  
 #Jacobi buf  
 Jacobi\_new = []  
 for row in range(0, N \* 5):  
 Jacobi\_new.append([])  
 for column in range(0, N \* 5):  
 Jacobi\_new[row].append(0)  
  
 #print("Start nuton\_iter")  
 while (Now\_eps > eps):  
 timer = time.time()  
 Count\_Iterions += 1  
 Jacobi\_buf = Jacobi.copy()  
 #print("Df(0) = ", Def\_Function[0])  
 #Subs jacobi  
 if Multi == 1 and N != 1:  
 Data = []  
 for row in range(0, N \* 5):  
 for column in range(0, N \* 5):  
 Data.append((Jacobi\_buf[row][column], Symbol\_data))  
 with mp.Pool(60) as p:  
 p\_result = p.map(Sub\_data, Data)  
 for row in range(0, N \* 5):  
 for column in range(0, N \* 5):  
 Jacobi\_new[row][column] = p\_result[row \* (N \* 5) + column]  
 #print('Subs jacodi\_multi = Time %.6f' % (time.time() - timer))  
 else:  
 for row in range(0, N \* 5):  
 for column in range(0, N \* 5):  
 Jacobi\_new[row][column] = Jacobi\_buf[row][column].subs(Symbol\_data)  
 #print('Subs jacodi\_not multi = Time %.6f' % (time.time() - timer))  
  
 Jackobi\_Matrix = sym.Matrix(Jacobi\_new)  
 Jacobi\_Invariant = Jackobi\_Matrix.inv()  
  
 timer = time.time()  
 Result\_new,Step\_x = Get\_New\_iterarion\_Nuton(Def\_Function, Jacobi\_Invariant, Result\_now, Step\_x)  
 #print('Get\_New\_iterarion = Time %.6f' % (time.time() - timer))  
  
 print("Result\_new = ",Result\_new)  
 np.array(Result\_new).astype(np.float64)  
  
 Max\_eps = 0  
 for i in range(0, N):  
 New\_Eps[i] = abs(Result\_new[i] - Result\_now[i])  
 if (New\_Eps[i] > Max\_eps):  
 Max\_eps = New\_Eps[i]  
  
 Last\_Result = Result\_now  
 Now\_eps = Max\_eps  
 Result\_now = Result\_new  
  
 All\_Results.append(Result\_now)  
 if Count\_Iterions > 25:  
 Step\_x = Step\_x / 5  
 Result\_now = Last\_Result  
 if All\_Results[Count\_Iterions - 3][0] < All\_Results[Count\_Iterions - 2][0] < \  
 All\_Results[Count\_Iterions - 1][0]:  
 if Step\_x != 1:  
 Step\_x \*= 5  
 Result\_now = Result\_new  
 if All\_Results[Count\_Iterions - 3][0] > All\_Results[Count\_Iterions - 2][0] > \  
 All\_Results[Count\_Iterions - 1][0]:  
 if Step\_x != 1:  
 Step\_x \*= 5  
 Result\_now = Result\_new  
 if Count\_Iterions > 50:  
 Now\_eps = 0  
 #print("\n Iteration =", Count\_Iterions, " and eps =", Now\_eps, " \n")  
  
 return Result\_now  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Main code  
 timer = time.time()  
  
 # Collect W-V func  
 w\_vals = Get\_w\_coefs()  
 u\_vals = Get\_u\_coefs()  
 v\_vals = Get\_v\_coefs()  
 PsiX\_vals = Get\_PsiX\_coefs()  
 PsiY\_vals = Get\_PsiY\_coefs()  
  
 #print("Vals")  
 #print(w\_vals)  
 #print(u\_vals)  
 #print(v\_vals)  
 #print(PsiX\_vals)  
 #print(PsiY\_vals)  
  
 W\_function = Get\_W\_function\_vals(w\_vals)  
 U\_function = Get\_U\_function\_vals(u\_vals)  
 V\_function = Get\_V\_function\_vals(v\_vals)  
 PsiX\_function = Get\_PsiX\_function\_vals(PsiX\_vals)  
 PsiY\_function = Get\_PsiY\_function\_vals(PsiY\_vals)  
  
 #print("Func")  
 #print(W\_function)  
 #print(U\_function)  
 #print(V\_function)  
 #print(PsiX\_function)  
 #print(PsiY\_function)  
  
 # Заготовочка array's  
 Max\_W\_values = [0] \* 3  
 Max\_Sigmas\_values = [0] \* 3  
 Es\_main = [0] \* Count\_Es  
 W\_middle\_values = []  
 W\_middle\_2\_values = []  
 Q\_values = []  
  
 # Заготовочка for results  
 Sigma\_x\_Function = 5 \* Xx  
 Sigma\_y\_Function = 5 \* Xx  
 Tay\_xy\_Function = 5 \* Xx  
 Q\_function = 5 \* Xx  
  
 # Sigmas  
 U\_function\_buf = U\_function.copy()  
 V\_function\_buf = V\_function.copy()  
 W\_function\_buf = W\_function.copy()  
 PsiX\_function\_buf = PsiX\_function.copy()  
 PsiY\_function\_buf = PsiY\_function.copy()  
  
  
 # Es main  
 z\_num = 0  
 z\_val = -h/2  
 KSI\_1 = ksi\_1(PsiX\_function,PsiY\_function)  
 KSI\_2 = ksi\_2(PsiX\_function,PsiY\_function)  
 KSI\_12 = ksi\_12(PsiX\_function,PsiY\_function)  
 TETTA\_1 = Tetta\_1(W\_function,U\_function)  
 TETTA\_2 = Tetta\_2(W\_function,V\_function)  
 F\_EX = e\_x(U\_function,V\_function,W\_function,TETTA\_1)  
 F\_EY = e\_y(U\_function,V\_function,W\_function,TETTA\_2)  
 F\_XY = y\_xy(U\_function,V\_function,W\_function,TETTA\_1, TETTA\_2)  
 F\_XY\_part =y\_xy\_part(U\_function,V\_function,W\_function,TETTA\_1, TETTA\_2)  
  
  
 F\_EX\_z = e\_x\_z(F\_EX, KSI\_1)  
 F\_EY\_z = e\_y\_z(F\_EY, KSI\_2)  
 F\_XY\_z = y\_xy\_z(F\_XY, KSI\_12)  
  
 #z\_val = 0  
 type\_z = 1  
 if Change == 0:  
 Sigma\_x\_Function = Get\_Sigma\_x\_Orto(U\_function\_buf, V\_function\_buf, W\_function\_buf, E1, nu\_12, nu\_21,z\_val, PsiX\_function\_buf, PsiY\_function\_buf)  
 Sigma\_y\_Function = Get\_Sigma\_y\_Orto(U\_function\_buf, V\_function\_buf, W\_function\_buf, E2, nu\_12, nu\_21, z\_val, PsiX\_function\_buf, PsiY\_function\_buf)  
 Tay\_xy\_Function = Get\_Sigma\_tay\_Orto(U\_function\_buf, V\_function\_buf, W\_function\_buf, G12, z\_val, PsiX\_function\_buf, PsiY\_function\_buf)  
 if Change == 2 or Change == 3 or Change == 1:  
 if type\_z == 1:  
 Sigma\_x\_Function = Get\_Sigma\_x\_Izo(E1, nu\_12,nu\_21, z\_val, F\_EX\_z, F\_EY\_z, KSI\_1, KSI\_2)  
 Sigma\_y\_Function = Get\_Sigma\_y\_Izo(E1, nu\_12,nu\_21, z\_val, F\_EX\_z, F\_EY\_z, KSI\_1, KSI\_2)  
 Tay\_xy\_Function = Get\_Sigma\_tay\_Izo(G12, nu\_12, z\_val, F\_XY\_z, KSI\_12)  
 if type\_z == 0:  
 Sigma\_x\_Function = Get\_Sigma\_x\_Izo(E1, nu\_12,nu\_21, z\_val, F\_EX, F\_EY, KSI\_1, KSI\_2)  
 Sigma\_y\_Function = Get\_Sigma\_y\_Izo(E1, nu\_12,nu\_21, z\_val, F\_EX, F\_EY, KSI\_1, KSI\_2)  
 Tay\_xy\_Function = Get\_Sigma\_tay\_Izo(G12, nu\_12, z\_val, F\_XY, KSI\_12)  
  
 #print("Sigma\_x\_Function = ", Sigma\_x\_Function)  
 #print("Sigma\_y\_Function = ", Sigma\_y\_Function)  
 #print("Tay\_xy\_Function = ", Tay\_xy\_Function)  
 integral\_type = 1  
 part = 1  
 if integral\_type == 1:  
 if Change == 0:  
 Es\_main[0] = N\_x\_Orto(z\_num, U\_function, V\_function, W\_function, E1, nu\_12, nu\_21) \* e\_x(U\_function, V\_function, W\_function)  
 Es\_main[1] = N\_y\_Orto(z\_num, U\_function, V\_function, W\_function, E1, nu\_12, nu\_21) \* e\_y(U\_function, V\_function, W\_function)  
 Es\_main[2] = N\_xy\_Orto(z\_num, U\_function, V\_function, W\_function, G12) \* y\_xy(U\_function, V\_function,W\_function)  
 Es\_main[3] = M\_x\_Orto(W\_function, E1, nu\_12, nu\_21) \* ksi\_1(PsiX\_function,PsiY\_function) \  
 + M\_y\_Orto(W\_function,E2,nu\_12,nu\_21) \* ksi\_2(PsiX\_function, PsiY\_function)  
 Es\_main[4] = 2 \* M\_xy\_Orto(W\_function, G12) \* ksi\_12(PsiX\_function, PsiY\_function)  
 Es\_main[5] = Q\_x(PsiX\_function, PsiY\_function, G13, W\_function, U\_function)  
 Es\_main[6] = Q\_y(PsiX\_function, PsiY\_function, G23, W\_function, U\_function)  
 if Change == 2 or Change == 3 or Change ==1:  
 if part ==1:  
 NX\_I\_part = N\_x\_Izo\_part(E1, nu\_12, nu\_21, F\_EX, F\_EY)  
 NY\_I\_part = N\_y\_Izo\_part(E1, nu\_12, nu\_21, F\_EX, F\_EY)  
 #NXY\_I = N\_xy\_Izo(G12, nu\_12,F\_XY)  
 NXY\_I\_part = N\_xy\_Izo(G12, nu\_12, F\_XY\_part)  
  
 MX\_I = M\_x\_Izo(E1, nu\_12,nu\_21,KSI\_1,KSI\_2)  
 MY\_I = M\_y\_Izo(E1,nu\_12,nu\_21,KSI\_1,KSI\_2)  
 MXY\_I = M\_xy\_Izo(G12, nu\_12,KSI\_12)  
  
 Q\_X = Q\_x(PsiX\_function,G13, TETTA\_1)  
 Q\_Y = Q\_y(PsiY\_function, G23, TETTA\_2)  
  
 Es\_main[0] = NX\_I\_part[0] \* F\_EX  
 #print("Es\_main[0] = ",Es\_main[0])  
 Es\_main[1] = NX\_I\_part[1] \* F\_EX  
 #print("Es\_main[1] = ", Es\_main[1])  
 Es\_main[2] = NY\_I\_part[0] \* F\_EY  
 #print("Es\_main[2] = ", Es\_main[2])  
 Es\_main[3] = NY\_I\_part[1] \* F\_EY  
 #print("Es\_main[3] = ", Es\_main[3])  
 Es\_main[4] = NXY\_I\_part[0]\*F\_XY  
 #print("Es\_main[4] = ", Es\_main[2])  
 Es\_main[5] = MX\_I \* KSI\_1 + MY\_I \* KSI\_2 + (MXY\_I + MXY\_I) \* KSI\_12  
 #print("Es\_main[5] = ", Es\_main[5])  
 Es\_main[6] = Q\_X \* (PsiX\_function - TETTA\_1) + Q\_Y \* (PsiY\_function - TETTA\_2)  
 #print("Es\_main[6] = ", Es\_main[6])  
 Es\_main[8] = NXY\_I\_part[1] \* F\_XY  
 #print("Es\_main[8] = ", Es\_main[8])  
 Es\_main[9] = NXY\_I\_part[2] \* F\_XY\_part[0]  
 #print("Es\_main[9] = ", Es\_main[9])  
 Es\_main[10] = NXY\_I\_part[2] \* F\_XY\_part[1]  
 #print("Es\_main[10] = ", Es\_main[10])  
 Es\_main[11] = NXY\_I\_part[2] \* F\_XY\_part[2]  
 #print("Es\_main[11] = ", Es\_main[11])  
 if part == 0:  
 NX\_I = N\_x\_Izo(E1, nu\_12, nu\_21, F\_EX, F\_EY)  
 NX\_I\_part = N\_x\_Izo\_part(E1, nu\_12, nu\_21, F\_EX, F\_EY)  
 NY\_I = N\_y\_Izo(E1, nu\_12, nu\_21, F\_EX, F\_EY)  
 NXY\_I = N\_xy\_Izo(G12, nu\_12, F\_XY)  
  
 MX\_I = M\_x\_Izo(E1, nu\_12, KSI\_1, KSI\_2)  
 MY\_I = M\_y\_Izo(E1, nu\_12, KSI\_1, KSI\_2)  
 MXY\_I = M\_xy\_Izo(G12, nu\_12, KSI\_12)  
  
 Q\_X = Q\_x(PsiX\_function, G13, TETTA\_1)  
 Q\_Y = Q\_y(PsiY\_function, G23, TETTA\_2)  
  
 Es\_main[0] = NX\_I \* F\_EX  
 print("Es\_main[0] = ", Es\_main[0])  
 Es\_main[1] = NY\_I \* F\_EY  
 print("Es\_main[1] = ", Es\_main[1])  
 Es\_main[2] = (1 / 2) \* (NXY\_I + NXY\_I) \* F\_XY  
 print("Es\_main[2] = ", Es\_main[2])  
 Es\_main[3] = MX\_I \* KSI\_1 + MY\_I \* KSI\_2  
 print("Es\_main[3] = ", Es\_main[3])  
 Es\_main[4] = (MXY\_I + MXY\_I) \* KSI\_12  
 print("Es\_main[4] = ", Es\_main[4])  
 Es\_main[5] = Q\_X \* (PsiX\_function - TETTA\_1)  
 print("Es\_main[5] = ", Es\_main[5])  
 Es\_main[6] = Q\_Y \* (PsiY\_function - TETTA\_2)  
 print("Es\_main[6] = ", Es\_main[6])  
 print("New\_main")  
  
  
 W\_val = []  
 U\_val = []  
 V\_val = []  
 PsiX\_val = []  
 PsiY\_val = []  
 W\_values = []  
  
 Es\_main\_buf = Es\_main.copy()  
  
 if One\_point == 1:  
 if Type\_Resolve == 1:  
 print("Start Nuton\_point")  
 W\_values = Nuton\_byPoint(Es\_main\_buf,W\_function)  
 if Type\_Resolve == 2:  
 print("Start Bubnov")  
 W\_values = Bubnov\_byPoint(NX\_I, NY\_I, NXY\_I, MX\_I, MY\_I, MXY\_I, Q\_X, Q\_Y, TETTA\_1, TETTA\_2, 3.14)  
 else:  
 if Type\_Resolve == 1:  
 print("Start Nuton\_loop")  
 W\_values = Nuton\_Loop(Es\_main\_buf,W\_function,Sigma\_x\_Function,Sigma\_y\_Function,Tay\_xy\_Function)  
 if Type\_Resolve == 2:  
 print("Start Bubnov")  
 W\_values = Bubnov\_Loop(NX\_I, NY\_I, NXY\_I, MX\_I, MY\_I, MXY\_I, Q\_X, Q\_Y, TETTA\_1, TETTA\_2, 3.14)  
  
 print("W\_vals")  
 print(W\_values)  
 spend\_time = time.time() - timer  
 print("Prog work by = ", spend\_time)  
  
 print(W\_values)  
 for i in range(1, N + 1):  
 W\_val.append(W\_values[i - 1])  
 U\_val.append(W\_values[i - 1 + N])  
 V\_val.append(W\_values[i - 1 + 2 \* N])  
 PsiX\_val.append(W\_values[i - 1 + 3 \* N])  
 PsiY\_val.append(W\_values[i - 1 + 4 \* N])  
  
 Symbol\_Function = [0] \* N \* 5  
 # print( "A = ", a)  
 for i in range(0, N):  
 Symbol\_Function[i] = 'w' + str(i + 1)  
 Symbol\_Function[i + N] = 'u' + str(i + 1)  
 Symbol\_Function[i + N \* 2] = 'v' + str(i + 1)  
 for index in range(0, N):  
 W\_function = W\_function.subs(Symbol\_Function[index], W\_values[index])  
 U\_function = U\_function.subs(Symbol\_Function[index + N], W\_values[index + N])  
 V\_function = V\_function.subs(Symbol\_Function[index + N \* 2], W\_values[index + N \* 2])  
 #print("Results =")  
 #print(W\_function)  
 #print(U\_function)  
 #print(V\_function)  
 Draw\_3d\_W(W\_function,1,1)  
 Draw\_deformed\_plot(W\_function, U\_function, V\_function)  
  
 # Prints data :  
 #print("W values = ", W\_val)  
 #print("U values = ", U\_val)  
 #print("V values = ", V\_val)  
 #print("Psi X values = ", V\_val)  
 #print("Psi Y values = ", V\_val)  
 print('Time %.6f' % (time.time() - timer))