Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

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

Факультет строительный

Кафедра информационных технологий

**ОТЧЕТ ПО НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ**

**Линейно-упругая задача для стальной плиты**

Обучающийся Мельниченко Д. С.

Направление подготовки: 01.03.02 – Прикладная математика и информатика

Группа: ПМИб-3

Оценка

Дата

Преподаватель

Семенов А.А., заведующий каф. ИТ

*(подпись) (Ф.И.О., должность)*

Санкт-Петербург

2023 г.

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc132881556)

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc132881557)

[ОБЩИЕ ТЕОРЕТИЧЕСКИЕ МАТЕРИАЛЫ 4](#_Toc132881558)

[МАТЕМАТИЧЕСКАЯ МОДЕЛЬ 7](#_Toc132881559)

[РАСЧЕТЫ 8](#_Toc132881560)

[ЗАКЛЮЧЕНИЕ 18](#_Toc132881561)

[ПРИЛОЖЕНИЕ 19](#_Toc132881562)

# ПОСТАНОВКА ЗАДАЧИ

Рассматривается линейно-упругая задача для плиты с разными видами закрепления, длиной *a* [м], шириной *b* [м] и толщиной *h* [м] находящейся под действием распределенной нагрузки *q* [Мпа] . Стоит задача рассмотреть три варианта заделки: жесткой, шарнирной, комбинированной включающей в себя часть жесткой, и часть шарнирной заделки.

# ОБЩИЕ ТЕОРЕТИЧЕСКИЕ МАТЕРИАЛЫ

Выведем уравнение равновесия пластины толщиной *h*, находящейся под действием поперечной нагрузки *q*(*x, y*). Срединную поверхность (плоскость) пластины примем за координатную и сведем трехмерную задачу к двумерной относительно деформации срединной поверхности, введя следующие ограничения:

1. Пластина допускает малые прогибы, поэтому соотношения между деформациями и перемещениями (геометрические соотношения) будут линейными.
2. Справедлива гипотеза прямой нормали, согласно которой первоначально прямолинейный и нормальный к срединной поверхности элемент остается при деформировании пластины прямолинейным и нормальным. При этом перемещения в слое, отстоящем на *z* от срединной поверхности, имеют вид
3. Для тонких пластин (*h / a*< 1 / 20) пренебрегается вертикальными напряжениями .
4. Будем считать, что в пластине под действием нагрузки возникают только изгибные деформации.
5. Материал пластины изотропный и упругий (связь между напряжениями и деформациями линейная). Основные соотношения деформирования пластины состоят из геометрических и физических соотношений и функционала полной энергии деформации.

При введенных предположениях геометрические соотношения в срединной поверхности будут иметь вид (связь между деформациями и перемещениями):



С учетом ограничения 4 (учитываются только изгибные деформации):





Так как модель основывается на минимизации функционала полной энергии, то необходимо привести его формулу (c учетом наших предположений):



Здесь выражения для моментов получаются интегрированием компонент напряжений по толщине пластины:





Функционал полной потенциальной энергии деформации равен разности потенциальной энергии системы и работы внешних сил *А*:



Потенциальная энергия системы – это работа внутренних сил, которая в общем виде записывается как:



# МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Функционал полной энергии деформации балки при изгибе будет имеет вид



где



Данные значения подставляются в функционал:



# РАСЧЕТЫ

Таблица 1. Зависимость  от различных видов заделок и разного кол-ва *N*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Тип закрепления | *N* | Max (*W*) [м] | Max () [МПа] | Max () [МПа] | Max () [МПа] |
| 1 | Жесткая | 1 | 0.013411 | 26.2428 | 26.2428 | 7.1070 |
| 4 | 0.013412 | 19.4545 | 19.4545 | 9.6119 |
| 9 | 0.014391 | 20.3788 | 20.3788 | 10.4624 |
| 2 | Шарнирная | 1 | 0.043487 | 42.7942 | 42.7942 | 23.1107 |
| 4 | 0.043487 | 42.7942 | 42.7942 | 23.1107 |
| 3 | Комбинированная | 1 | 0.020239 | 24.4507 | 35.0967 | 10.7404 |
| 4 | 0.010389 | 10.9002 | 13.2687 | 5.8315 |
| 9 | 0.008249 | 9.1629 | 11.6164 | 4.4541 |

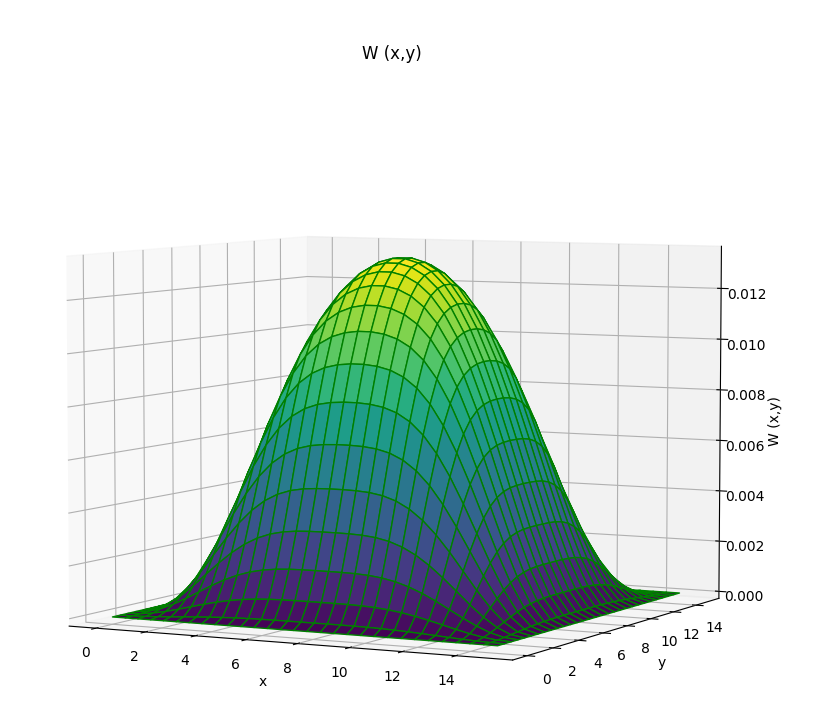


Рисунок 1. График *W (x,y)* для жесткой заделки при *N* = 4

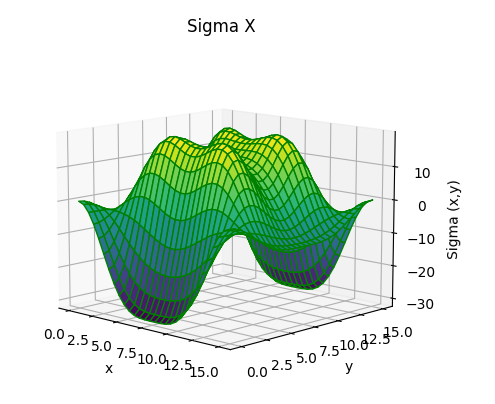


Рисунок 2. График  для жесткой заделки при *N* = 4

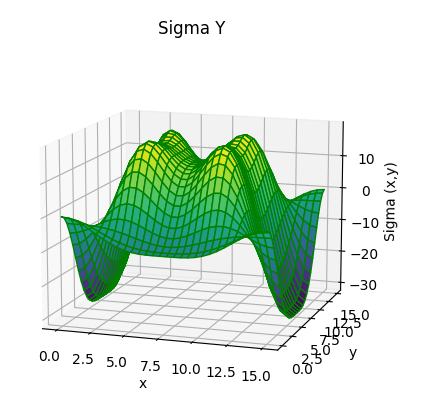


Рисунок 3. График  для жесткой заделки при *N* = 4

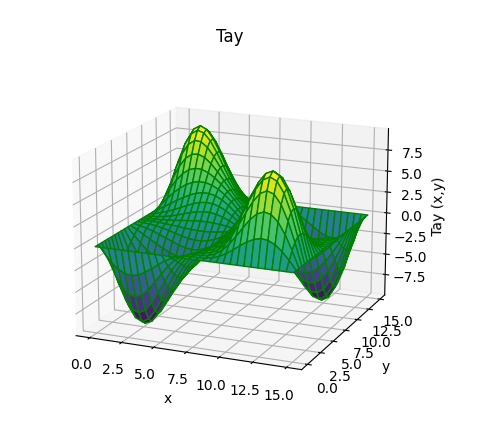


Рисунок 4. График  для жесткой заделки при *N* = 4

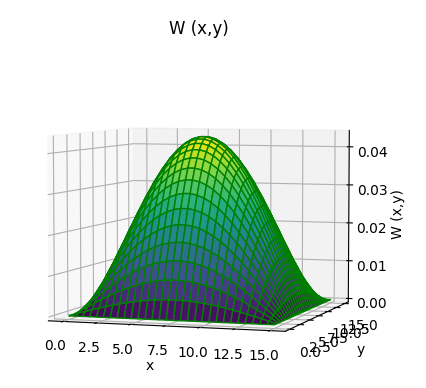
****

Рисунок 5. График *W (x,y)* для шарнирной заделки при *N* = 4

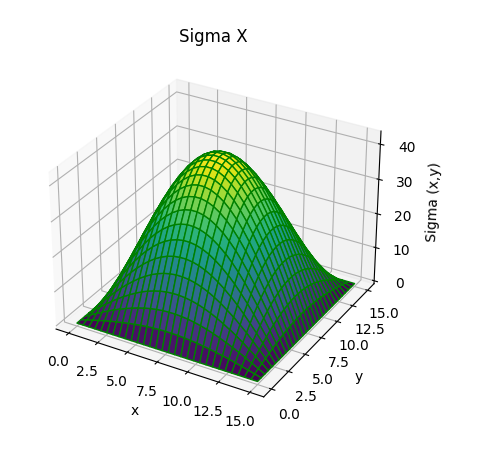
****

Рисунок 6. График  для шарнирной заделки при *N* = 4

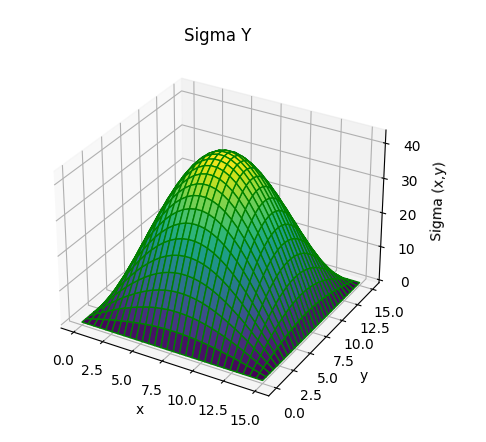
****

Рисунок 7. График  для шарнирной заделки при *N* = 4

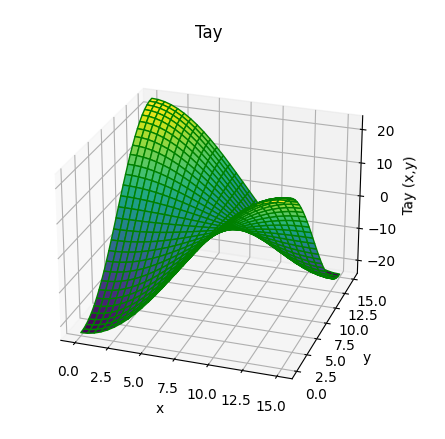
****

Рисунок 8. График  для шарнирной заделки при *N* = 4

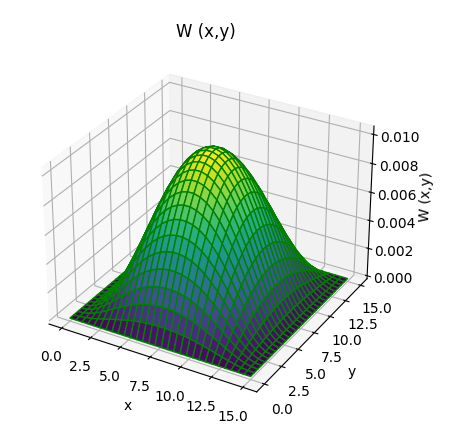
****

Рисунок 9. График *W (x,y)* для комбинированной заделки при *N* = 4

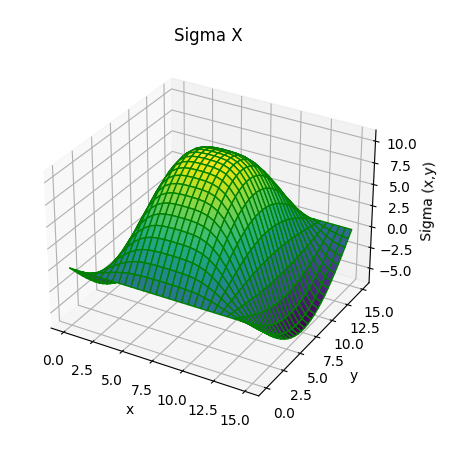
****

Рисунок 10. График  для комбинированной заделки при *N* = 4

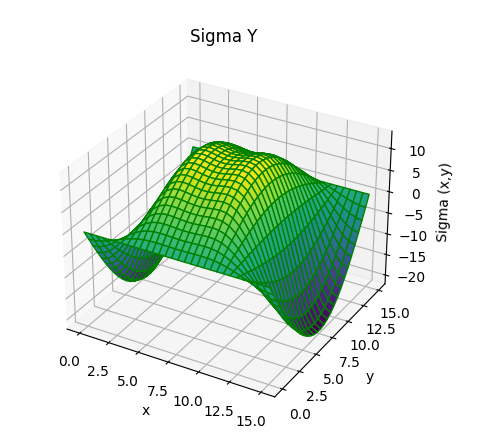
****

Рисунок 11. График  для комбинированной заделки при *N* = 4

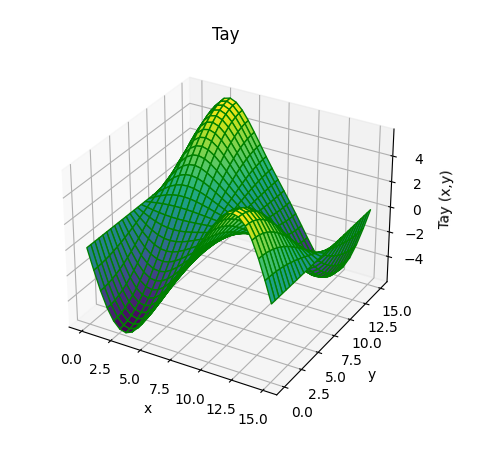
****

Рисунок 12. График  для комбинированной заделки при *N* = 4

# ЗАКЛЮЧЕНИЕ

Была рассмотрена линейно-упругая задача для плиты с различными видами закрепления. В каждом случае закрепления варьировалась длина (кол-во членов аппроксимирующей функции) и сравнивались полученные результаты.

В результате можно утверждать, что комбинированный метод наиболее эффективен, так как имеет наименьший прогиб (сравнивая максимальные прогибы при одинаковой нагрузке), а также время, затраченное на построение комбинированного случая, лишь немногим превосходило время на просчет других видов заделки.

# ПРИЛОЖЕНИЕ

Программа написана на языке программирования Python в среде PyCharm:

import sympy as sym  
from sympy import \*  
import numpy as np  
import math as m  
import matplotlib as mpl  
from mpl\_toolkits import mplot3d  
import matplotlib.pyplot as plt  
  
#Выбор типа функции  
Choose = 2  
Type = 2  
#DATA  
N=9  
E=2.1\*(10\*\*5)  
q\_T=1.34/100  
L=15  
h=0.15  
z\_val = h/2  
#graph points  
size\_Graph = 20  
Size = 30  
Max\_q\_T = 2  
A=L  
B=L  
x = Symbol('x')  
Yy = Symbol('y')  
Q\_\_T = q\_T  
Mm=10\*\*5  
eps = 0.001  
nu = 0.3  
#Settings to integral  
a = 0  
b = L  
  
#Создать w функцию  
def Draw\_3d\_W(dw):  
 x\_array = []  
 y\_array = []  
 z\_array = [0]\*Size  
 for i in range (0,Size):  
 z\_array[i] = [0]\*Size  
 step = A/(Size-1)  
 for i in range(0,Size):  
 x\_array.append(i\*step)  
 y\_array.append(i\*step)  
  
 for i in range(0,Size):  
 for j in range(0,Size):  
 z\_array[i][j] = (Get\_W\_Plane(x\_array[i],y\_array[j],dw))  
  
 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')  
 ax.set\_ylabel('y')  
 ax.set\_zlabel('W (x,y)')  
 ax.set\_title('W (x,y)')  
 plt.show()  
def Sigmas(dw,Coefs):  
 w =Get\_W\_To\_Sigmas(Coefs)  
 x = Symbol('x')  
 y = Symbol('y')  
 All\_w = 0  
 for i in range(0,N):  
 All\_w += w[i]  
  
 w\_for\_Sigma\_x = All\_w.diff(x)  
 w\_for\_Sigma\_x = w\_for\_Sigma\_x.diff(x)  
  
 w\_for\_Sigma\_y = All\_w.diff(y)  
 w\_for\_Sigma\_y = w\_for\_Sigma\_y.diff(y)  
  
 w\_for\_tay = All\_w.diff(x)  
 w\_for\_tay = w\_for\_tay.diff(y)  
  
 x\_array = []  
 y\_array = []  
 Sigma\_X = [0] \* Size  
 Sigma\_Y = [0] \* Size  
 Tay = [0] \* Size  
  
 for i in range(0, Size):  
 Sigma\_X[i] = [0] \* Size  
 Sigma\_Y[i] = [0] \* Size  
 Tay[i] = [0] \* Size  
  
 step = A / (Size - 1)  
 for i in range(0, Size):  
 x\_array.append(i \* step)  
 y\_array.append(i \* step)  
  
  
 Plot\_Sigmas\_X(Sigma\_X,x\_array,y\_array,w\_for\_Sigma\_x,dw,w\_for\_Sigma\_x,w\_for\_Sigma\_y,w\_for\_tay)  
 Plot\_Sigmas\_Y(Sigma\_X, x\_array, y\_array, w\_for\_Sigma\_y, dw,w\_for\_Sigma\_x,w\_for\_Sigma\_y,w\_for\_tay)  
 Plot\_Tay(Sigma\_X, x\_array, y\_array, w\_for\_tay, dw,w\_for\_Sigma\_x,w\_for\_Sigma\_y,w\_for\_tay)  
  
def Plot\_Sigmas\_X(Sigmas\_array,x\_array,y\_array,dw\_Sigmas,dw,dw\_X,dw\_Y,dw\_xy):  
 y = Symbol('y')  
 x = Symbol('x')  
 max = 0;  
 for W\_coefs in range(0, N):  
 #dw\_Sigmas = dw\_Sigmas.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_X = dw\_X.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_Y = dw\_Y.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_xy = dw\_xy.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
  
 for i in range(0,Size):  
 for j in range(0,Size):  
 Sigmas\_array[i][j] = (-z\_val\*E/(1 - nu\*\*2)\*(dw\_X + nu\*dw\_Y)).subs([(x,x\_array[i]),(y,y\_array[j])])  
 if Sigmas\_array[i][j] > max:  
 max =Sigmas\_array[i][j]  
  
 print("Max Sigma x =",max)  
  
 Sigmas\_array = np.array(Sigmas\_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, Sigmas\_array, cmap='viridis', edgecolor='green')  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('y')  
 ax.set\_zlabel('Sigma (x,y)')  
 ax.set\_title('Sigma X')  
 #ax.xlabel('x')  
 #ax.ylabel('y')  
 #ax.zlabel('Sigma(x,y)')  
 plt.show()  
def Plot\_Sigmas\_Y(Sigmas\_array, x\_array, y\_array, dw\_Sigmas, dw,dw\_X,dw\_Y,dw\_xy):  
 y = Symbol('y')  
 x = Symbol('x')  
 max = 0;  
 for W\_coefs in range(0, N):  
 # dw\_Sigmas = dw\_Sigmas.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_X = dw\_X.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_Y = dw\_Y.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_xy = dw\_xy.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
  
 for i in range(0, Size):  
 for j in range(0, Size):  
 Sigmas\_array[i][j] = (-z\_val\*E/(1 - nu\*\*2)\*(dw\_Y + nu\*dw\_X)).subs([(x, x\_array[i]), (y, y\_array[j])])  
 if Sigmas\_array[i][j] > max:  
 max =Sigmas\_array[i][j]  
  
 print("Max Sigma y =", max)  
 Sigmas\_array = np.array(Sigmas\_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, Sigmas\_array, cmap='viridis', edgecolor='green')  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('y')  
 ax.set\_zlabel('Sigma (x,y)')  
 ax.set\_title('Sigma Y')  
 plt.show()  
def Plot\_Tay(Sigmas\_array, x\_array, y\_array, dw\_Sigmas, dw,dw\_X,dw\_Y,dw\_xy):  
 y = Symbol('y')  
 x = Symbol('x')  
 max = 0;  
  
 for W\_coefs in range(0, N):  
 # dw\_Sigmas = dw\_Sigmas.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_X = dw\_X.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_Y = dw\_Y.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
 dw\_xy = dw\_xy.subs('w' + str(W\_coefs + 1), (dw[W\_coefs]))  
  
 for i in range(0, Size):  
 for j in range(0, Size):  
 Sigmas\_array[i][j] = (-z\_val\*E/(1 + nu)\*dw\_xy).subs([(x, x\_array[i]), (y, y\_array[j])])  
 if Sigmas\_array[i][j] > max:  
 max =Sigmas\_array[i][j]  
  
 print("Max tay =", max)  
 Sigmas\_array = np.array(Sigmas\_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, Sigmas\_array, cmap='viridis', edgecolor='green')  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('y')  
 ax.set\_zlabel('Tay (x,y)')  
 ax.set\_title('Tay')  
 plt.show()  
def Create\_w():  
 w\_coefs = []  
 # Add w1..n  
 for i in range(1, N + 1):  
 w\_coefs.append(Symbol('w' + str(i)))  
 return w\_coefs  
def Create\_w\_Plane():  
 w\_coefs = []  
 # Add w1..n  
 #Hard  
 for i in range(1, N + 1):  
 w\_coefs.append(Symbol('w' + str(i)))  
 return w\_coefs  
  
#return W  
def Get\_W(X,dw):  
 W\_result = 0.  
 for i in range(1, N + 1):  
 W\_result+= dw[i-1] \* sin((m.pi \* i \* X)/L)  
  
 return W\_result  
def Get\_W\_To\_Sigmas(Coef):  
 w = []  
 y=Symbol('y')  
 x=Symbol('x')  
 New\_n = N \*\* (1 / 2.0)  
 New\_n = int(New\_n)  
 # Hard  
 if Type == 0:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j - 1) \* New\_n + i - 1] \* (1 - cos(2 \* i \* y \* m.pi / L)) \* (1 - cos(2 \* j \* x \* m.pi / L)))  
 if Type == 1:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j - 1) \* New\_n + i - 1] \* sin(i \* y \* m.pi / L) \* sin(j \* x \* m.pi / L))  
 if Type == 2:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j - 1) \* New\_n + i - 1] \* (1 - cos(2 \* i \* y \* m.pi / L)) \* sin(j \* x \* m.pi / L))  
  
 return w  
def Get\_W\_Plane(X\_val,Y\_val,dw):  
 W\_result = 0.  
 New\_n = N \*\* (1 / 2.0)  
 New\_n = int(New\_n)  
  
  
 # Hard  
 if Type == 0:  
 for j in range(1, New\_n+1):  
 for i in range(1, New\_n+1):  
 W\_result+= dw[(j-1)\*New\_n + i-1] \* (1 - cos(2 \* i \* Y\_val \* m.pi / A)) \* (1 - cos(2 \* j \* X\_val \* m.pi / B))  
  
 if Type == 1:  
 for j in range(1, New\_n+1):  
 for i in range(1, New\_n+1):  
 W\_result+= dw[(j-1)\*New\_n + i-1] \* sin(i \* Y\_val \* m.pi / L) \* sin(j \* X\_val \* m.pi / L)  
  
 if Type == 2:  
 for j in range(1, New\_n+1):  
 for i in range(1, New\_n+1):  
 W\_result+= dw[(j-1)\*New\_n + i-1] \* (1 - cos(2 \* i \* Y\_val \* m.pi / L)) \* sin(j \* X\_val \* m.pi / L)  
  
 return W\_result  
  
def Es\_1\_function(x,y,n,l,Coef):  
 Pp = Symbol('pi')  
 Ll = Symbol('l')  
 w = []  
 w\_2 = []  
 w\_3 = []  
 New\_n = N \*\* (1 / 2.0)  
 New\_n = int(New\_n)  
  
 # Hard  
 if Type == 0:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* (1 - cos(2 \* i \* y \* m.pi / L)) \* (1 - cos(2 \* j \* x \* m.pi / L)))  
  
 if Type == 1:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* sin(i \* y \* m.pi / L) \* sin(j \* x \* m.pi / L))  
 if Type == 2:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* (1 - cos(2 \* i \* y \* m.pi / Ll)) \* sin(j \* x \* m.pi / L))  
  
 for i in range(1, n + 1):  
 w\_2.append(w[i - 1].diff(x))  
 for i in range(1, n + 1):  
 w\_3.append(w\_2[i - 1].diff(x))  
  
 return w\_3  
def Es\_2\_function(x,y,n,l,Coef):  
 Pp = Symbol('pi')  
 Ll = Symbol('l')  
 w = []  
 w\_2\_x = []  
 w\_2\_y = []  
 w\_3\_y = []  
 w\_3\_x = []  
 w\_res = []  
 New\_n = N \*\* (1 / 2.0)  
 New\_n = int(New\_n)  
 # Hard  
 if Type == 0:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \*(1 - cos(2 \* i \* y \* m.pi / L)) \* (1 - cos(2 \* j \* x \* m.pi / L)))  
 if Type == 1:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* sin(i \* y \* m.pi / L) \* sin(j \* x \* m.pi / L))  
 if Type == 2:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[i - 1] \* (1 - cos(2 \* i \* y \* m.pi / Ll)) \* sin(j \* x \* m.pi / L))  
  
 for i in range(1, n + 1):  
 w\_2\_x.append(w[i - 1].diff(x))  
 w\_2\_y.append(w[i - 1].diff(y))  
 for i in range(1, n + 1):  
 w\_3\_y.append(w\_2\_y[i - 1].diff(y))  
 w\_3\_x.append(w\_2\_x[i - 1].diff(x))  
 for i in range(1, n + 1):  
 w\_res.append(w\_3\_y[i-1] \* w\_3\_x[i-1])  
 return w\_res  
def Es\_3\_function(x,y,n,l,Coef):  
 Pp = Symbol('pi')  
 Ll = Symbol('l')  
 w = []  
 w\_2 = []  
 w\_3 = []  
 New\_n = N\*\*(1/2.0)  
 New\_n = int(New\_n)  
 #Hard  
 if Type == 0:  
 for j in range(1,New\_n+1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* ( 1 - cos(2\*i\*y\*m.pi/L)) \* ( 1 - cos(2\*j\*x\*m.pi/L)) )  
 if Type == 1:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* sin(i \* y \* m.pi / L) \* sin(j \* x \* m.pi / L))  
 if Type == 2:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[i - 1] \* (1 - cos(2 \* i \* y \* m.pi / Ll)) \* sin(j \* x \* m.pi / L))  
  
 for i in range(1, n + 1):  
 w\_2.append(w[i - 1].diff(y))  
 for i in range(1, n + 1):  
 w\_3.append(w\_2[i - 1].diff(y))  
 return w\_3  
def Es\_4\_function(x,y,n,l,Coef):  
 Pp = Symbol('pi')  
 Ll = Symbol('l')  
 w = []  
 w\_2 = []  
 w\_3 = []  
 New\_n = N \*\* (1 / 2.0)  
 New\_n = int(New\_n)  
 # Hard  
 if Type == 0:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* (1 - cos(2 \* i \* y \* m.pi / L)) \* (1 - cos(2 \* j \* x \* m.pi / L)))  
 if Type == 1:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* sin(i \* y \* m.pi / L) \* sin(j \* x \* m.pi / L))  
 if Type == 2:  
 for j in range(1, New\_n + 1):  
 for i in range(1, n + 1):  
 w.append(Coef[i - 1] \* (1 - cos(2 \* i \* y \* m.pi / L)) \* sin(j \* x \* m.pi / L))  
 for i in range(1, n + 1):  
 w\_2.append(w[i - 1].diff(x))  
 for i in range(1, n + 1):  
 w\_3.append(w\_2[i - 1].diff(y))  
 return w\_3  
def Es\_5\_function(x, y, n, l, Coef):  
 Pp = Symbol('pi')  
 Ll = Symbol('l')  
 w = []  
 New\_n = n \*\* (1 / 2.0)  
 New\_n = int(New\_n)  
 # Hard  
 if Type == 0:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* (1 - cos(2 \* i \* y \* m.pi / L)) \* (1 - cos(2 \* j \* x \* m.pi / L)))  
  
 if Type == 1:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[(j-1)\*New\_n + i-1] \* sin(i \* y \* m.pi / Ll) \* sin(j \* x \* m.pi / Ll))  
  
 if Type == 2:  
 for j in range(1, New\_n + 1):  
 for i in range(1, New\_n + 1):  
 w.append(Coef[i - 1] \* (1 - cos(2 \* i \* y \* m.pi / Ll)) \* sin(j \* x \* m.pi / Ll))  
  
 return w  
#Loop for Nuton  
def Get\_New\_iterarion(Function,Jackobi\_inv,W,a):  
 Def\_Function = [0] \* N  
  
 for i in range(0, N):  
 Def\_Function[i] = Function.diff('w' + str(i + 1))  
 for W\_coefs in range(0, N):  
 Def\_Function[i] = Def\_Function[i].subs('w' + str(W\_coefs + 1), (W[W\_coefs]))  
  
 Def\_Function = sym.Matrix(Def\_Function)  
 W = sym.Matrix(W)  
 W = W - a\*(Jackobi\_inv \* Def\_Function)  
 return W  
#Линейный варик  
def Lin\_Function\_Plane(w\_coefs,Es\_1,Es\_2,Es\_3,Es\_4,Es\_5,N):  
 Ee = Symbol('E')  
 Hh = Symbol('h')  
 Qq = Symbol('q')  
 Ll = Symbol('l')  
 dw = []  
  
 for i in range(1, N + 1):  
 buf = Es\_1.diff(w\_coefs[i - 1]) + Es\_2.diff(w\_coefs[i - 1]) + Es\_3.diff(w\_coefs[i - 1]) + Es\_4.diff(w\_coefs[i - 1]) - Es\_5.diff(w\_coefs[i - 1])  
 #Сокращение ~~0 коэф.  
 for j in range(1, N + 1):  
 if j != i:  
 buf = buf.subs('w' + str(j), 0)  
 buf = buf.subs([(Ee, E), (Hh, h), (Qq, Q\_\_T), (Ll, L), (pi, m.pi)])  
 dw.append(solve(buf)[0])  
  
 return dw  
  
#Создание апрк. функции  
w\_coefs = Create\_w\_Plane()  
  
#Создание частей Es  
Es\_1 = Es\_1\_function(x,Yy,N,L,w\_coefs)  
Es\_2 = Es\_2\_function(x,Yy,N,L,w\_coefs)  
Es\_3 = Es\_3\_function(x,Yy,N,L,w\_coefs)  
Es\_4 = Es\_4\_function(x,Yy,N,L,w\_coefs)  
Es\_5 = Es\_5\_function(x,Yy,N,L,w\_coefs)  
  
  
#Соед.Частей Es  
  
Es\_1\_result = 0  
Es\_2\_result = 0  
Es\_3\_result = 0  
Es\_4\_result = 0  
Es\_5\_result = 0  
for i in range (0,N):  
 Es\_1\_result += Es\_1[i]  
 Es\_2\_result += Es\_2[i]  
 Es\_3\_result += Es\_3[i]  
 Es\_4\_result += Es\_4[i]  
 Es\_5\_result += Es\_5[i]  
  
  
#Приведение к нужному виду  
Es\_1 = Es\_1\_result\*\*2  
Es\_2 = Es\_2\_result  
Es\_3 = Es\_3\_result\*\*2  
Es\_4 = Es\_4\_result\*\*2  
Es\_5 = Es\_5\_result  
  
Es\_1 = integrate(Es\_1, (x, a, A))  
Es\_2 = integrate(Es\_2, (x, a, A))  
Es\_3 = integrate(Es\_3, (x, a, A))  
Es\_4 = integrate(Es\_4, (x, a, A))  
Es\_5 = integrate(Es\_5, (x, a, A))  
  
Es\_1 = integrate(Es\_1, (Yy, a, B))  
Es\_2 = integrate(Es\_2, (Yy, a, B))  
Es\_3 = integrate(Es\_3, (Yy, a, B))  
Es\_4 = integrate(Es\_4, (Yy, a, B))  
Es\_5 = integrate(Es\_5, (Yy, a, B))  
  
#Создание символов/Коэф.Интегралов  
Ee = Symbol('E')  
Hh = Symbol('h')  
Qq = Symbol('q')  
Ll = Symbol('l')  
D = ( Ee\*(Hh\*\*3) )/ (12 \* (1 - nu\*\*2) )  
  
Es\_1 \*= D/2  
Es\_2 \*= nu\*D  
Es\_3 \*= D/2  
Es\_4 \*= (1 - nu)\*D  
Es\_5 \*= Qq  
  
#Массив производных  
#Es=y\_1  
dw = []  
dw\_2 = []  
  
#Данные для графика  
x\_now\_1=0  
X\_count\_1 = L/0.5  
X\_step\_1 = L/X\_count\_1  
X\_for\_graph\_1 =[]  
W\_graph =[]  
  
print("Hi")  
  
  
dw = Lin\_Function\_Plane(w\_coefs,Es\_1,Es\_2,Es\_3,Es\_4,Es\_5,N)  
Result = Get\_W\_Plane(A/2,B/2, dw)  
print(Result)  
print(dw)  
print(w\_coefs)  
Draw\_3d\_W(dw)  
Sigmas(dw,w\_coefs)