Санкт-Петербургский политехнический университет

Высшая школа теоретической механики, ФизМех

Направление подготовки

«01.03.03 Механика и математическое моделирование»

Индивидуальное задание № 1

тема "Метод конечных элементов. Расчет статического прогиба балки Бернулли-Эйлера"

дисциплина "Вычислительная механика"

Вариант 2

Выполнила студент гр. 5030103/90301 Бенюх М. А.

Преподаватель: Е.Ю. Витохин

Санкт-Петербург

2022

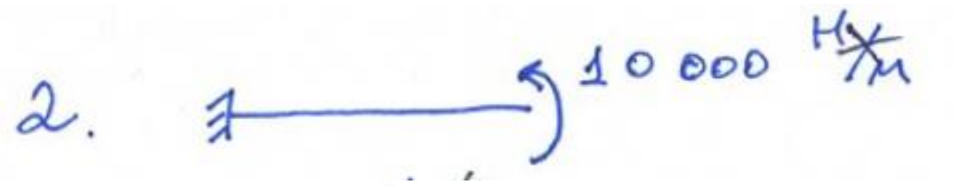
Содержание:

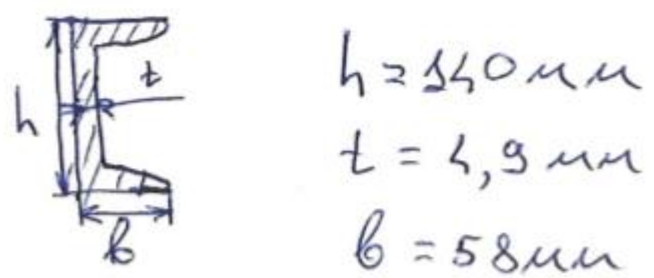
1. Формулировка задачи.

Произвести расчет статического прогиба балки Бернулли-Эйлера. Закрепить крайне левый конец балки заделкой. Требуется определить перемещения в балке фермы и усилия в стержнях. В качестве сечения использовать швеллер с высотой $h=$ 140 мм, толщиной стенки $d=$ 4.9 мм и шириной полки $b=$ 58 мм.

Таблица 1. Параметры задачи

Параметр	Значение
Коэффициент Пуассона	0.35
Модуль Юнга **E** (Па)	$2\ 10^{11}$
Момент **M** (Н/м)	$10\ 10^{3}$





2. Алгоритм метода.

Введем систему координат.
Предполагается, что при изгибе балка не выходит из плоскости (X, Y). Поле перемещений балки разделим на продольную (перемещения u(x, y)) и поперечную (прогиб v(x, y)) составляющие. Важно отметить, что компоненты поля перемещений связаны между собой следующим соотношением:

$$u(x,y) = -y\frac{\partial v}{\partial x} = -y\theta, \qquad \text{где } \theta - \text{угол поворота сечения.}$$

Продольное напряженно-деформированного состояние описывается следующим образом:

$$\varepsilon_{xx} = \varepsilon = \frac{\partial u}{\partial x} = -y\frac{\partial \theta}{\partial x} = -y\varkappa, \qquad \text{где } \varkappa - \text{кривизна балки.}$$

Используем закон Гука:

$$\sigma_{xx} = \sigma = E\varepsilon = -y\varkappa E$$

Изгибающий момент:

$$M_z = -\int_S y\sigma dS = \varkappa E\int_S y^2\sigma dS = J\varkappa E$$

$$J = J_{zz} = \int_S y^2 dS - \text{момент инерции сечения}$$

Момент инерции вычисляется по формуле:

$$J = \frac{bh^3 - (b-t)(h-2t)^3}{12}$$

Рассчитывать изгиб будем исходя из вариационного принципа минимума потенциальной энергии, для чего построим функционал потенциальной энергии следующего вида:

$$\Pi = \Lambda - W,$$

где Λ — энергия деформации, W — работа внешних сил

$$\Lambda = \frac{1}{2}EJ\int_l \varkappa^2 dl$$

$$W = W_c + W_v + W_s$$

В условиях нашей задачи $W_v = W_s = 0 -$ работы поверхностных и объемных сил.

$$W_c = P_c\overline{u^e} - \text{работа сосредоточенных сил}$$

Будем рассматривать балочный конечный элемент. Вектор перемещений элемента записывается следующим образом:

$$\{u^e\}^T = \{v_i \quad \theta_i \quad v_j \quad \theta_j\}$$

Для перемещений принимается кубическая аппроксимация.

$$u^e = A + Bx + Cx^2 + Dx^3$$

Для перехода к дискретной модели используются функции формы узлов, для которых также принимается кубическая аппроксимация.

$$u^e = N_i v_i + N_i^\theta \theta_i + N_j v_j + N_j^\theta \theta_j = [N]\{u^e\}$$

Для изопараметрического элемента функции формы имеют вид:

$$N_i = \frac{1}{4}(1-\xi)^2(2+\xi) \qquad N_i^\theta = \frac{l}{8}(1-\xi)^2(1+\xi)$$
$$N_j = \frac{1}{4}(1+\xi)^2(2-\xi) \qquad N_j^\theta = -\frac{l}{8}(1+\xi)^2(1-\xi)$$

Вычислим кривизну:

$$\varkappa = \frac{d^2 v(x)}{dx^2} = [B]\{u^e\}, \qquad \text{где } [B] - \text{матрица градиентов}$$

$$[B] = \frac{1}{l}\left[\frac{6\xi}{l}, \quad 3\xi - 1, \quad -\frac{6\xi}{l}, \quad 3\xi + 1\right]$$

Тогда энергия деформации будет задана выражением:

$$\Lambda = \frac{1}{2} EJ\{u^e\}^T \int_{-1}^{1} \frac{l}{2}[B]^T[B]\, d\xi$$

Работа внешних сил:

$$W = \int_l \{u^e\}^T[N]^T\{P_c\}dl + \{u^e\}^T\{P_c\}$$

Приравняем:

$$\frac{1}{2} EJ \int_{-1}^{1} \frac{l}{2}[B]^T[B]\, d\xi = \int_l [N]^T\{P_c\}dl + \{P_c\}$$

Подставим матрицу [B] и выделим коэффициент:

$$[k^e] = \frac{EJ}{l^3} \cdot \begin{bmatrix} 12 & 6l & -12 & 6l \\ 6l & 2l^2 & -6l & 2l^2 \\ -12 & -6l & 12 & -6l \\ 6l & 2l^2 & -6l & 2l^2 \end{bmatrix} - \text{матрица жесткости элемента}$$

$$\{f^e\} = \int_l [N]^T\{P_c\}dl + \{P_c\}$$

Подводя итог, можем выделить отсюда СЛАУ:

$$[k^e]\{u^e\} = \{f^e\},$$

Переходя ко всей балке:

$$[K]\{U\} = \{F\},$$

$$[K] = \sum_e [k^e], \qquad [F] = \sum_e [f^e]$$

3. Результаты.

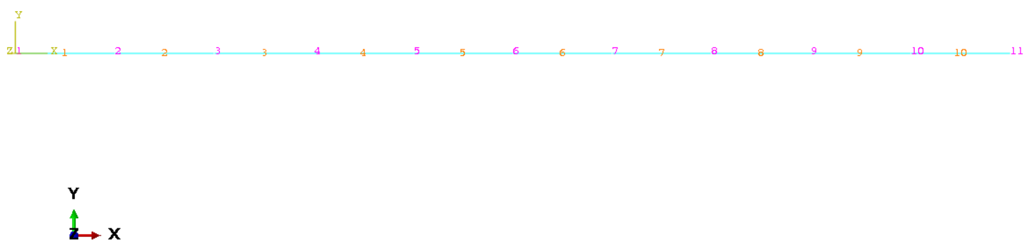


Рис.2. Номера узлов и элементов балки

3.1. Результаты работы в Abaqus

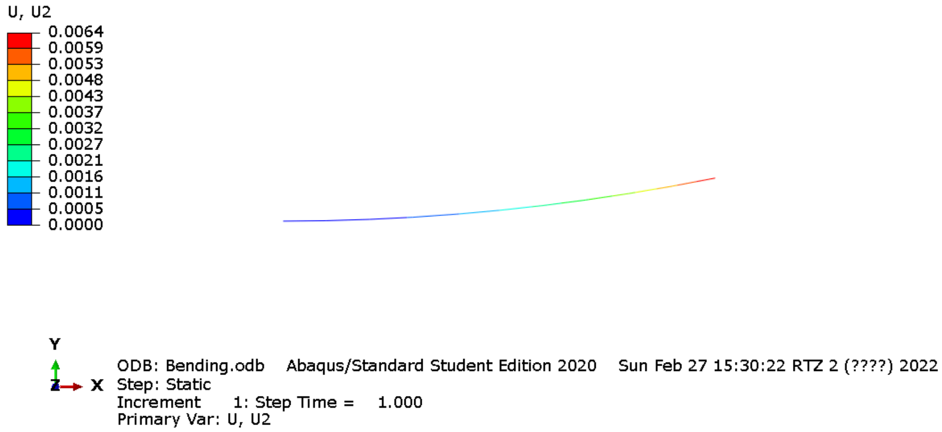


Рис.3. Поле перемещений по оси OY (м)

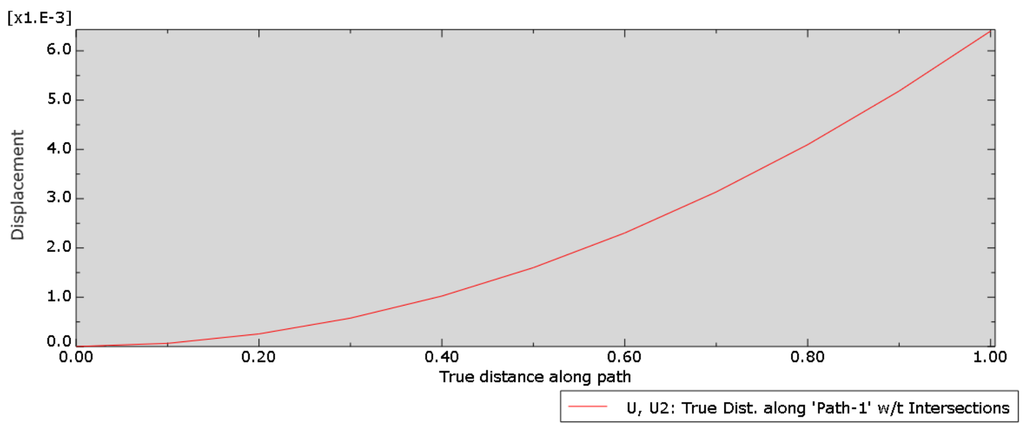


Рис.4. Перемещения

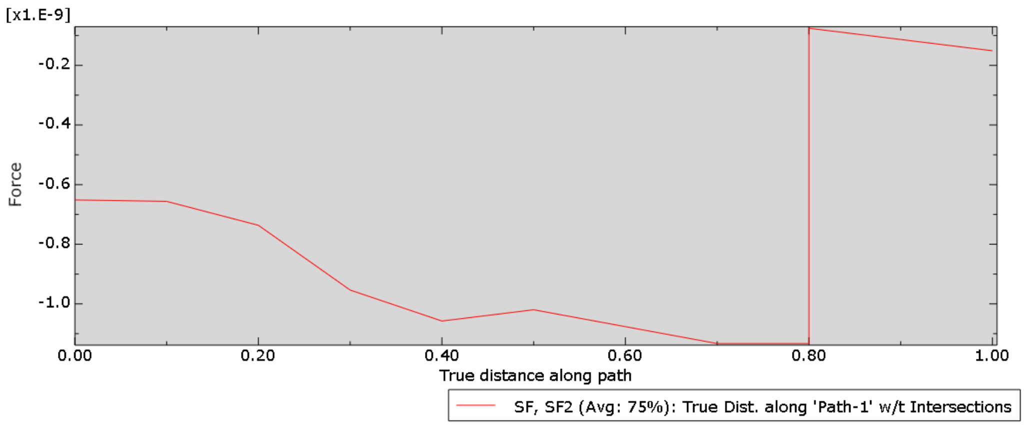


Рис.5. Усилия

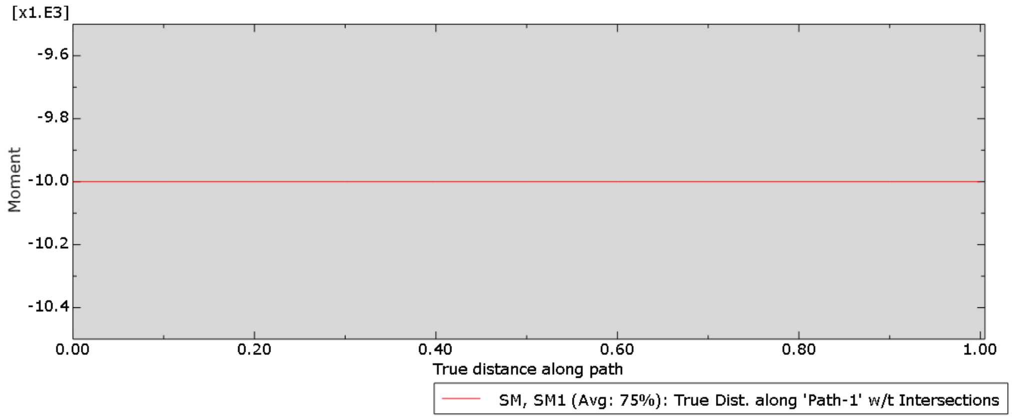


Рис 6. Моменты

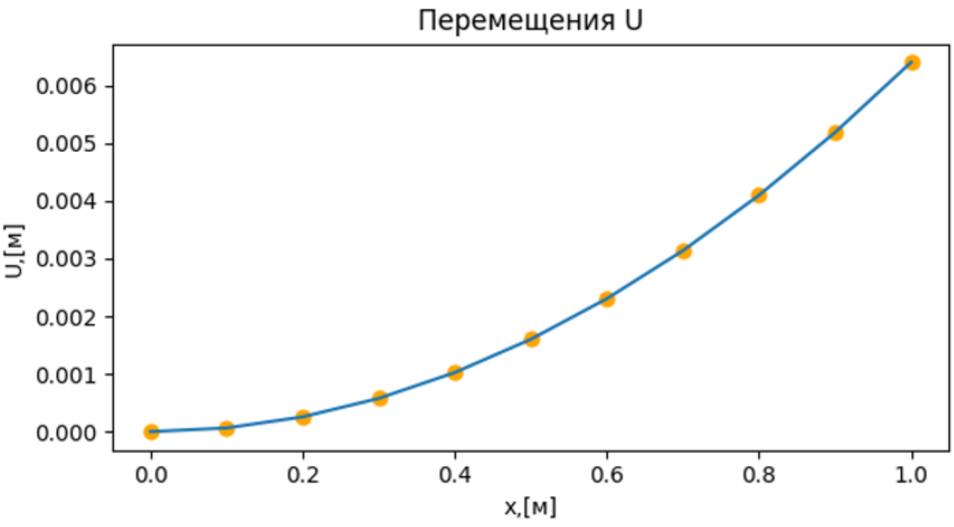


Рис.7. Перемещения

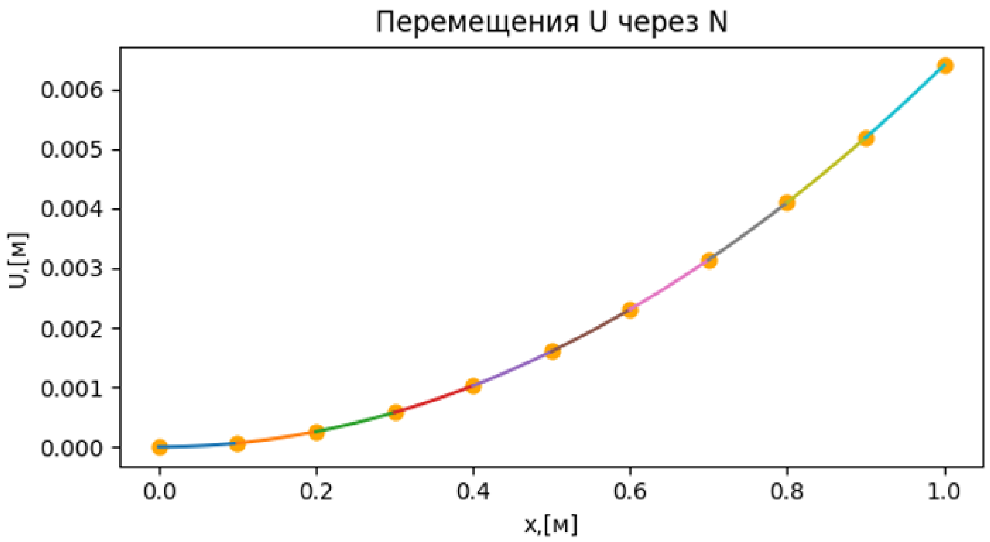


Рис.10. Прогиб

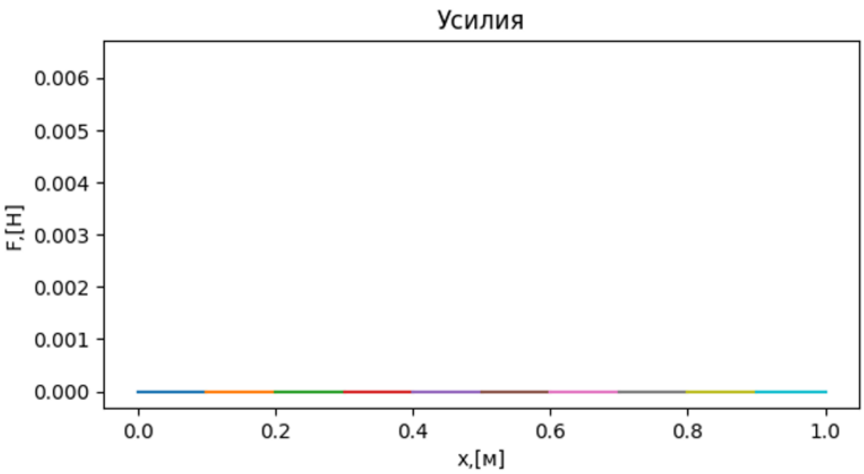


Рис.8. Усилия

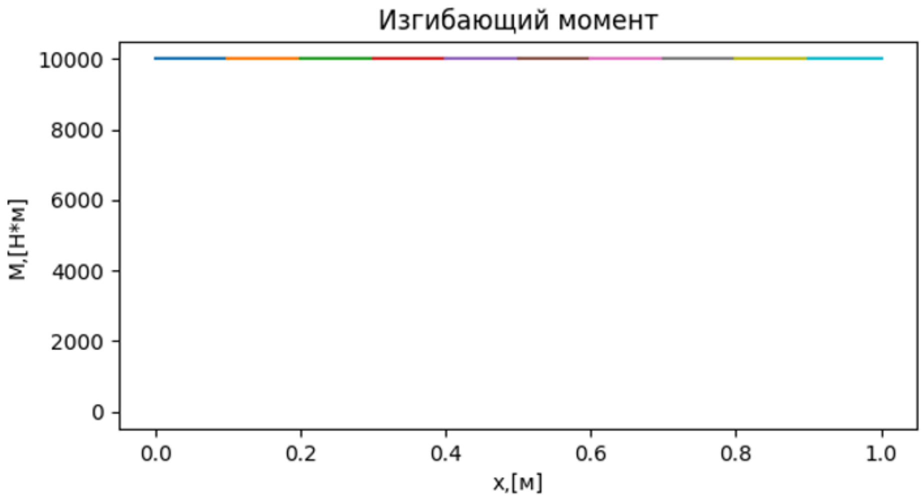


Рис 9. Моменты

3.2. Сравнение результатов

Перемещения, м,(U2)

X, м	Abaqus, Y	Python, Y
0	0	0.00000000e+00
0.1	6.40102E-05	6.40188471e-05
0.2	0.000256041	2.56075389e-04
0.3	0.000576092	5.76169624e-04
0.4	0.00102416	1.02430155e-03
0.5	0.00160026	1.60047118e-03
0.6	0.00230437	2.30467850e-03
0.7	0.0031365	3.13692351e-03
0.8	0.00409665	4.09720622e-03
0.9	0.00518483	5.18552662e-03
1	0.00640102	6.40188471e-03

Моменты, Н*м (SM1)

X , м	Abaqus, Y	Python, Y
0	10000	10000.
0.1	10000	10000.
0.2	10000	10000.
0.3	10000	10000.
0.4	10000	10000.
0.5	10000	10000.
0.6	10000	10000.
0.7	10000	10000.
0.8	10000	10000.
0.9	10000	10000.

1	10000	10000.

	Усилия, Н (SF2)	
Координата X, м	Abaqus, Y	Python, Y
0	-6.51528E-10	-2.87797564e-10
0.1	-6.56249E-10	0.00000000e+00
0.2	-7.3651E-10	1.43898782e-10
0.3	-9.53686E-10	0.00000000e+00
0.4	-1.05755E-09	2.01458294e-09
0.5	-1.01978E-09	0.00000000e+00
0.6	-1.07644E-09	1.72678538e-09
0.7	-1.13309E-09	0.00000000e+00
0.8	-1.13309E-09	1.72678538e-09
0.9	-7.55395E-11	-1.15119025e-09
1	-1.13309E-10	6.90714153e-09

Заключение

В рамках данной задачи с помощью метода конечных элементов были получены усилия, моменты, прогибы. В ходе работы проведен расчет статического прогиба балки Бернулли-Эйлера в Abaqus и Python. Полученные результаты совпадают с точностью до 7 знака.

Код программы

```
class Balcka:
    def __init__(self, x, l, E, rho, h, w, Mass_node, Mass_Element, P):
        print("__init__ Balcka")
        self.x = x
        self.l = l
        self.E = E
        self.rho = rho
        self.g = 9.8
        self.P = P
        self.Mass_node = Mass_node
        self.Mass_Element = Mass_Element
        self.type = self.Shveller(0.140, 0.058, 0.0049, 0.0049, 4)

        self.S = self.type.S
        self.k_e = self.CreateMatrix_k_e()
        self.getK_global()
        self.getF_global()


    def J(self):
        return self.type.I_x

    def N_i(self, eta):
        return 1 / 4 * ((1 - eta) ** 2) * (2 + eta)

    def d_N_i(self, eta):
        return (3 * (-1 + eta ** 2)) / 4

    def dd_N_i(self, eta):
        return (3 * eta) / 2

    def ddd_N_i(self, eta):
        return 3 / 2

    def N_i_theta(self, eta):
        return 1 / 8 * self.l * (1 + eta) * ((1 - eta) ** 2)

    def d_N_i_theta(self, eta):
        return (self.l * (-1 - 2 * eta + 3 * eta ** 2)) / 8

    def dd_N_i_theta(self, eta):
        return (self.l * (-1 + 3 * eta)) / 4

    def ddd_N_i_theta(self, eta):
        return (self.l * 3) / 4

    def N_j(self, eta):
        return 1 / 4 * (2 - eta) * ((1 + eta) ** 2)

    def d_N_j(self, eta):
        return (-3 * (-1 + eta ** 2)) / 4

    def dd_N_j(self, eta):
        return (-3 * eta) / 2

    def ddd_N_j(self, eta):
        return (-3) / 2

    def N_j_theta(self, eta):
        return -1 / 8 * self.l * (1 - eta) * ((1 + eta) ** 2)
```

```
    def d_N_j_theta(self, eta):
        return (self.l * (-1 + 2 * eta + 3 * eta ** 2)) / 8

    def dd_N_j_theta(self, eta):
        return (self.l * (1 + 3 * eta)) / 4

    def ddd_N_j_theta(self, eta):
        return (self.l * 3) / 4

    def CreateMatrix_B(self, eta):
        B = np.zeros((1, 4))
        B[0, 0] = 6 * eta / self.l
        B[0, 1] = 3 * eta - 1
        B[0, 2] = -6 * eta / self.l
        B[0, 3] = 3 * eta + 1
        B = B / self.l
        return B

    def B0(self, eta):
        return 6 * eta / self.l / self.l

    def B1(self, eta):
        return (3 * eta - 1) / self.l

    def B2(self, eta):
        return -6 * eta / self.l / self.l

    def B3(self, eta):
        return (3 * eta + 1) / self.l

    def CreateMatrix_k_e(self):
        k_e = np.zeros((4, 4))
        k_e[0, :] = [12, 6 * self.l, -12, 6 * self.l]
        k_e[1, :] = [6 * self.l, 4 * (self.l ** 2), -6 * self.l, 2 * (self.l
** 2)]
        k_e[2, :] = [-12, -6 * self.l, 12, - 6 * self.l]
        k_e[3, :] = [6 * self.l, 2 * (self.l ** 2), -6 * self.l, 4 * self.l
** 2]
        k_e = k_e * self.E * self.J() / (self.l ** 3)
        print("k_e")
        return k_e

    def CreateRow_f_e(self, eta):
        f_e = np.zeros((1, 4))
        # print(f_e)
        N = self.CreateColumn_N(eta)
        # P = self.P.transpose()
        # N.transpose() * P +self.l / 2 * P * np.matrix([[1], [self.l / 6],
[1], [-self.l / 6]]).transpose()+
        f_e = N.transpose() * P + self.l / 2 * P * np.matrix(
            [[1], [self.l / 6], [1], [-self.l / 6]]).transpose() + self.l / 2
* self.rho * self.S * self.g * np.matrix(
            [[1], [self.l / 6], [1], [-self.l / 6]]).transpose()
        print(f_e)
        return f_e

    def CreateColumn_N(self, eta):
        N = np.zeros((4, 1))
        N[:, 0] = [self.N_i(eta), self.N_i_theta(eta), self.N_j(eta),
self.N_j_theta(eta)]
        return N

    def CreateColumn_dd_N(self, eta):
```

```
        N = np.zeros((4, 1))
        N[:, 0] = [self.dd_N_i(eta), self.dd_N_i_theta(eta),
self.dd_N_j(eta), self.dd_N_j_theta(eta)]
        return N

    def Eta(self, x):
        return 2 * x / self.l - 1

    def getK_global(self):
        K = np.zeros((2 * len(self.Mass_node), 2 * len(self.Mass_node)))
        for i in range(len(self.Mass_Element)):
            nodes = self.Mass_Element[i, :]
            for k in range(2):
                for j in range(2):
                    K[2 * nodes[0, k], 2 * nodes[0, j]] += self.k_e[2 * k, 2
* j]
                    K[(2 * nodes[0, k]) + 1, (2 * nodes[0, j]) + 1] +=
self.k_e[(2 * k) + 1, (2 * j) + 1]
                    K[2 * nodes[0, k], (2 * nodes[0, j]) + 1] += self.k_e[2 *
k, (2 * j) + 1]
                    K[(2 * nodes[0, k]) + 1, 2 * nodes[0, j]] += self.k_e[(2
* k) + 1, 2 * j]
        return K

    def getF_global(self):
        X = [0, 1 / 10, 2 / 10, 3 / 10, 4 / 10, 5 / 10, 6 / 10, 7 / 10, 8 /
10, 9 / 10, 10 / 10]
        F = 0
        F2 = self.P
        F = self.P
        return F

    def setGuToK(self, GU):
        K_new = self.getK_global()
        for i in range(len(K_new)):
            for j in range(len(GU)):
                if (i == 2 * (GU[j] - 1)):
                    K_new[i, :] = 0
                    K_new[:, i] = 0
                    K_new[i + 1, :] = 0
                    K_new[:, i + 1] = 0
                    K_new[i, i] = 1
                    K_new[i + 1, i + 1] = 1

        return K_new

    def Solve(self, GU):

        K_new = self.getK_global()
        K_gu = self.setGuToK(GU)
        F = self.getF_global().transpose()
        print('\n'.join(' '.join(str(col) for col in row) for row in K_gu))
        print("det", np.linalg.det(K_gu))
        U = solve(K_gu, F)
        [X, Y] = self.getDeff(U)
        [X1, U_new] = self.getUfromN(U, 0)
        [Xm, U_m] = self.getMfromNandU(U, 0)
        [Xm2, U_m2] = self.getMfromBandU(U, 0)
        stresses = X * self.E
        forces = stresses * self.type.S
        plt.subplot(2, 2, 1)
        self.Graph(X1, U_new, True, 1)
        plt.xlabel('x,[м]')
```

```
        plt.ylabel('U,[м]')
        plt.title('Перемещения U через N')
        plt.subplot(2, 2, 2)
        plt.plot(np.linspace(0, self.x, 11), X)
        plt.scatter(np.linspace(0, self.x, 11), X, color='orange', s=40,
marker='o')
        plt.xlabel('x,[м]')
        plt.ylabel('U,[м]')
        plt.title('Перемещения U')
        plt.subplot(2, 2, 3)
        self.Graph(Xm, U_m, False, 1)
        # plt.plot(np.linspace(0, self.x, 11), stresses)
        # plt.scatter(np.linspace(0, self.x, 11), stresses, color='orange',
s=40, marker='o')
        plt.xlabel('x,[м]')
        plt.ylabel('M,[Н*м]')
        plt.title('Изгибающий момент')
        plt.subplot(2, 2, 4)
        # plt.plot(np.linspace(0, self.x, 11), forces)
        # plt.scatter(np.linspace(0, self.x, 11), forces, color='orange',
s=40, marker='o')
        self.Graph(Xm2, U_m2, False, 1)
        plt.xlabel('x,[м]')
        plt.ylabel('F,[Н]')
        plt.title('Усилия')
        plt.show()

        return U

    def getDeff(self, U):
        X = np.zeros((int(len(U) / 2)))
        Y = np.zeros((int(len(U) / 2)))
        k = 0
        j = 0
        for i in range(len(U)):
            # print(U[i])
            if (i % 2) != 0:
                Y[k] = U[i]
                k += 1
            else:
                X[j] = U[i]
                j += 1
        self.X = X
        self.Y = Y
        print(Y, "Y")
        print(X, "X")
        return [X, Y]

    def getUfromN(self, U, number):
        X = np.zeros((int(len(U) / 2 - 1), 20))
        U_new = np.zeros((int(len(U) / 2 - 1), 20))
        for i in range(int(len(U) / 2 - 1)):
            X[i] = np.linspace(i * self.l, (i + 1) * self.l, 20)
            # print(np.linspace(i*self.l, (i+1)*self.l, 20))
        for i in range(int(len(U) / 2 - 1)):
            U_new[i] = self.N_i(self.Eta(X[number])) * U[2 * i] +
self.N_i_theta(self.Eta(X[number])) * U[
                2 * i + 1] + self.N_j(self.Eta(X[number])) * U[2 * i + 2] +
self.N_j_theta(self.Eta(X[number])) * U[
                           2 * i + 3]

        return [X, U_new]
```

```
    def getMfromNandU(self, U, number):  # M
        X = np.zeros((int(len(U) / 2 - 1), 20))
        U_new = np.zeros((int(len(U) / 2 - 1), 20))
        for i in range(int(len(U) / 2 - 1)):
            X[i] = np.linspace(i * self.l, (i + 1) * self.l, 20)
        for i in range(int(len(U) / 2 - 1)):
            U_new[i] = self.dd_N_i(self.Eta(X[number])) * U[2 * i] +
self.dd_N_i_theta(self.Eta(X[number])) * U[
                2 * i + 1] + self.dd_N_j(self.Eta(X[number])) * U[2 * i + 2]
+ self.dd_N_j_theta(self.Eta(X[number])) * \
                       U[
                           2 * i + 3]

        print(self.E * self.J() * 4 * U_new / (self.l ** 2), "U_M")
        return [X, self.E * self.J() * 4 * U_new / (self.l ** 2)]

    def getMfromBandU(self, U, number):  # F
        X = np.zeros((int(len(U) / 2 - 1), 20))
        U_new = np.zeros((int(len(U) / 2 - 1), 20))
        for i in range(int(len(U) / 2 - 1)):
            X[i] = np.linspace(i * self.l, (i + 1) * self.l, 20)
        for i in range(int(len(U) / 2 - 1)):
            U_new[i] = self.ddd_N_i(self.Eta(X[number])) * U[2 * i] +
self.ddd_N_i_theta(self.Eta(X[number])) * U[
                2 * i + 1] + self.ddd_N_j(self.Eta(X[number])) * U[2 * i + 2]
+ self.ddd_N_j_theta(
                self.Eta(X[number])) * U[
                           2 * i + 3]
        print(self.E * self.J() * 8 * U_new / (self.l ** 3), "U_F")
        return [X, self.E * self.J() * 8 * U_new / (self.l ** 3)]

    def Graph(self, X, U, var, koef):
        for i in range(len(X)):
            plt.plot(X[i], koef * U[i])
        if var:
            plt.scatter(np.linspace(0, self.x, 11), koef * self.X,
color='orange', s=40, marker='o')
        else:
            plt.scatter(np.linspace(0, self.x, 11), koef * self.X,
color='white', s=40, marker='o')


if __name__ == '__main__':
    E = 2 * (10 ** 11)  # сталь
    rho = 7700
    M = 10000
    l = 0.1
    x = 1
    w = 0.05
    h = 0.1
    Mass_node = np.matrix(
        [[0, 0], [0.1, 0], [0.2, 0], [0.3, 0], [0.4, 0], [0.5, 0], [0.6, 0],
[0.7, 0], [0.8, 0],
         [0.9, 0], [1, 0]])
    Mass_Element = np.matrix(
        [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [8,
9], [9, 10]])
    P = np.zeros((1, len(Mass_Element) * 2 + 2))
    P[0, len(Mass_Element) * 2 + 1] = M
    print(np.matrix([[1], [1], [1], [1]]))

    aaa = Balcka(x, l, E, rho, h, w, Mass_node, Mass_Element, P)
```

```
    aaa.CreateMatrix_B(1)
    U = aaa.Solve([1])
```