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 ЛАБОРАТОРНОЙ РАБОТЕ**

**Нелинейно-упругая задача для балки Эйлера-Бернулли**

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](#_Toc131074963)

[ВВЕДЕНИЕ 3](#_Toc131074964)

[МАТЕМАТИЧЕСКАЯ МОДЕЛЬ 4](#_Toc131074965)

[1. Основные теоретические материалы 4](#_Toc131074966)

[РАСЧЕТЫ 6](#_Toc131074967)

[ЗАКЛЮЧЕНИЕ 7](#_Toc131074968)

[ПРИЛОЖЕНИЕ 8](#_Toc131074969)

# ВВЕДЕНИЕ

Чтобы описать реальный физический процесс математическими зависимостями используются разные приемы и методы, которые исходят из природы процесса исследования. Пожалуй, основной метод построения математических моделей основан на применении фундаментальных законов природы, таких как сохранение энергии, массы вещества, импульса, числа частиц и др. Для построения математических моделей может применяться метод, основанный на вариационных принципах. Также для построения математических моделей может применяться принцип аналогий, иерархический подход, подход, основанный на приравнивании к нулю проекций всех силовых факторов (внутренних и внешних) по соответствующим направлениям осей координат.

Более широкими возможностями обладает математическое моделирование как метод исследования различных процессов путем описания их функционирования с помощью математических соотношений. Таким образом, при изучении процессов методом математического моделирования в первую очередь необходимо построить математическое описание изучаемого процесса. Математическая модель реального процесса – некоторый математический объект, поставленный в соответствие данному физическому процессу, т. е. математическое описание физического процесса с помощью алгебраических, дифференциальных, интегральных и других уравнений. Эти уравнения обычно выражают законы сохранения основных физических величин (энергии, количества движения, массы и др.) и связывают характеристики процесса с параметрами соответствующей системы, исходной информацией и начальными условиями.

# МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

# Основные 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:



Исходя из принятых предположений, функционал принимает вид



# РАСЧЕТЫ

Численные данные используемые при написании программы:

*l*=15[м] – длина балки,

*h* = 0.15[м] – толщина балки,

*E*=2.1·10^5[МПа] – модуль Юнга

*q*=1.34·10^5[МПа] – равномерно распределенная нагрузка.

Изображение выглядит как снимок экрана, текст, линия, График

Автоматически созданное описание

Рис. 2. Полученный в процессе график прогиба балки при N=3  
Оранжевая линия – линейный случай,  
Синяя линий – нелинейный случай

|  |  |  |
| --- | --- | --- |
| Зависимость: | W(x) при x= *l*/2 – линейная | W(x) при x= *l*/2 – нелинейная |
| N=1 | 0.150130398229424 | 0.154963506447133 |
| N=3 | 0.149512577660579 | 0.154659874622456 |
| N=5 | 0.149560619388012 | 0.154707599718522 |

Таблица 1 – Зависимость значений *W(x)* в середине балки от *N* и выбранного типа уравнения

# ЗАКЛЮЧЕНИЕ

Была рассмотрена нелинейно-упругая задача для балки с шарнирными закреплениями, находящейся под действием распределенной нагрузки *q(x)* [МПа]. Был найден прогиб балки и построен график зависимости прогиба от нагрузки.

Проанализировав полученные значения, можно сделать вывод, что при увеличении кол-ва функций, входящих в аппроксимирующую функцию, увеличивается и точность решения. Задав необходимые требования к точности решения, можно без проблем получить необходимое приближение к реальным значениям.

Начиная с некоторой нагрузки, можно наблюдать, что график нелинейной функции начинает идти “зигзагом”. Данная проблема, является проблемой математической модели, построенной посредством нелинейной функции.

# ПРИЛОЖЕНИЕ

import sympy as sym  
from sympy import \*  
import numpy as np  
import math as m  
import matplotlib as mpl  
import matplotlib.pyplot as plt  
  
#Выбор типа функции  
Choose =1  
#DATA  
N=3  
#graph points  
Size = 30  
Max\_q\_T = 2  
L=15  
x = Symbol('x')  
E=2.1\*(10\*\*5)  
q\_T=1.34/100  
Q\_\_T = q\_T  
h=0.15  
Mm=10\*\*5  
eps = 0.001  
#Settings to integral  
a = 0  
b = L  
  
#Создать w функцию  
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  
#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  
#Get Jakobi  
def Get\_Jacobian(Function, Result\_w):  
 Jacobian = [0]\*N  
 Def\_Function = [0]\*N  
  
 for i in range(0,N):  
 Jacobian[i] = [0]\*N  
 Def\_Function[i] = Function.diff('w' + str(i+1))  
  
 for row in range(0,N):  
 for column in range(0,N):  
 Jacobian[row][column] = Def\_Function[row].diff('w' + str(column+1))  
 for W\_coefs in range(0,N):  
 Jacobian[row][column] = Jacobian[row][column].subs('w' + str(W\_coefs+1), (Result\_w[W\_coefs]))  
  
 return Jacobian  
#Collect Es  
def function\_w(x,n,l,Coef):  
 Pp = Symbol('pi')  
 Ll = Symbol('l')  
 w=[]  
 w\_2=[]  
 w\_3 = []  
 for i in range(1,n+1):  
 w.append(Coef[i-1]\*sin(m.pi\*x\*i/Ll))  
 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 function\_w4(x,n,l,Coef):  
 Pp = Symbol('pi')  
 Ll = Symbol('l')  
 w = []  
 w\_2 = []  
 w\_3 = []  
 for i in range(1, n + 1):  
 w.append(Coef[i - 1] \* sin(m.pi \* x \* i / Ll))  
 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 fuction\_q(x,n,l,Coef):  
 Pp = Symbol('pi')  
 Ll = Symbol('l')  
 w = []  
 w\_2 = []  
 w\_3 = []  
 for i in range(1, n + 1):  
 w.append(Coef[i - 1] \* sin(m.pi \* x \* i / 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\_Loop(w\_coefs,y\_1,q\_1,fig, axs):  
 Ee = Symbol('E')  
 Hh = Symbol('h')  
 Qq = Symbol('q')  
 Ll = Symbol('l')  
 W\_Result = [0] \* (Size + 2)  
  
 W\_graph = []  
 Q\_now = 0.00001  
 Q\_step = q\_T/(Size/Max\_q\_T)  
 q\_for\_graph = []  
 # Цикл по разным q  
 for j in range(1, Size + 2):  
 dw = []  
 print("q\_now\_Linear = ", Q\_now)  
 for i in range(1, N + 1):  
 buf = y\_1.diff(w\_coefs[i - 1]) - q\_1.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\_now), (Ll, L), (pi, m.pi)])  
 dw.append(solve(buf)[0])  
 W\_graph.append(Get\_W(L / 2,dw))  
 print(Get\_W(L / 2,dw))  
 Q\_now += Q\_step  
 q\_for\_graph.append(Q\_now)  
 W\_Result.append(dw)  
  
 axs[0].plot(W\_graph, q\_for\_graph)  
 plt.show()  
  
 return W\_Result  
def Lin\_Function(w\_coefs,y\_1,q\_1,Q\_\_T):  
  
 Ee = Symbol('E')  
 Hh = Symbol('h')  
 Qq = Symbol('q')  
 Ll = Symbol('l')  
 W\_Result = []  
  
 # Цикл по разным q  
  
 dw = []  
 for i in range(1, N + 1):  
 buf = y\_1.diff(w\_coefs[i - 1]) - q\_1.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  
#Не линейный варик  
def Ne\_Lin\_Function\_Loop(w\_coefs,y\_1,q\_1,yn\_1,fig, axs):  
 Ee = Symbol('E')  
 Hh = Symbol('h')  
 Qq = Symbol('q')  
 Ll = Symbol('l')  
 W\_Result = [0]\*(Size+2)  
 W\_Result[0] = [0] \* N  
 Q\_now = 0.00001  
 Q\_step = q\_T/(Size/Max\_q\_T)  
 q\_for\_graph =[]  
 #Цикл по разным q  
  
 q\_for\_graph.append(Q\_now)  
 for j in range(1,Size+2):  
 W\_Result[j]=W\_Result[j-1]  
 print("q\_now\_Not\_Linear = ", Q\_now)  
  
  
 Buf\_Function = y\_1 + yn\_1 - q\_1  
 Main\_Function = Buf\_Function.subs([(Ee, E), (Hh, h), (Qq, Q\_now), (Ll, L), (pi, m.pi)])  
 W\_Result[j] = Nuton\_Iter(Main\_Function,eps, W\_Result[j], w\_coefs)  
  
 Q\_now += Q\_step  
 q\_for\_graph.append(Q\_now)  
 print(Get\_W(L / 2,W\_Result[j]))  
  
 W\_graph = []  
 for i in range(0,Size+2):  
 W\_graph.append(Get\_W(L/2,W\_Result[i]))  
  
 axs[0].plot(W\_graph,q\_for\_graph)  
 return W\_Result  
def Ne\_Lin\_Function(w\_coefs,y\_1,q\_1,yn\_1,Q\_\_T):  
 Ee = Symbol('E')  
 Hh = Symbol('h')  
 Qq = Symbol('q')  
 Ll = Symbol('l')  
  
 W\_Result = [0]\*N  
  
 Buf\_Function = y\_1 + yn\_1 - q\_1  
 Main\_Function = Buf\_Function.subs([(Ee, E), (Hh, h), (Qq, Q\_\_T), (Ll, L), (pi, m.pi)])  
  
 W\_Result = Nuton\_Iter(Main\_Function, eps, W\_Result, w\_coefs)  
  
 return W\_Result  
#Nut iter  
def Nuton\_Iter(F,eps,w0,w\_coefs):  
 Now\_eps = 1  
 All\_Results = []  
 Res\_now = []  
 Res\_now = w0  
 Res\_Last\_now = Res\_now  
 Res\_new = []  
 Count\_Iter = 0  
 Check\_Loop =1  
 Loop = 0  
  
 New\_Eps = [0]\*N  
  
 while(Now\_eps > eps):  
 Count\_Iter += 1  
 Max\_eps = 0  
  
 Jacobi = Get\_Jacobian(F,Res\_now)  
 Jackobi\_Matrix = sym.Matrix(Jacobi)  
 Jacobi\_Invariant = Jackobi\_Matrix.inv()  
  
 Res\_new = Get\_New\_iterarion(F,Jacobi\_Invariant,Res\_now,Check\_Loop)  
 np.array(Res\_new).astype(np.float64)  
  
 for i in range(0,N):  
 New\_Eps[i] = abs(Res\_new[i] - Res\_now[i])  
 if(New\_Eps[i] > Max\_eps):  
 Max\_eps = New\_Eps[i]  
  
 Res\_Last\_now = Res\_now  
 Now\_eps = Max\_eps  
 Res\_now = Res\_new  
 All\_Results.append(Res\_now)  
  
 if Count\_Iter > 10:  
 Check\_Loop = Check\_Loop/10  
 Res\_now = Res\_Last\_now  
 if All\_Results[Count\_Iter-3][0] < All\_Results[Count\_Iter-2][0] < All\_Results[Count\_Iter-1][0]:  
 if Check\_Loop != 1:  
 Check\_Loop \*= 10  
 Res\_now = Res\_new  
 if All\_Results[Count\_Iter-3][0] > All\_Results[Count\_Iter-2][0] > All\_Results[Count\_Iter-1][0]:  
 if Check\_Loop != 1:  
 Check\_Loop \*= 10  
 Res\_now = Res\_new  
  
 if Count\_Iter > 50:  
 Now\_eps = 0  
  
 return Res\_now  
  
#Создание апрк. функции  
w\_coefs = Create\_w()  
  
#Создание частей Es  
y = function\_w(x,N,L,w\_coefs)  
q = fuction\_q(x,N,L,w\_coefs)  
y\_neLin = function\_w4(x,N,L,w\_coefs)  
  
#Соед.Частей Es  
y\_result=0  
q\_result=0  
y\_neLin\_result=0  
for i in range (0,N):  
 y\_result += y[i]  
 q\_result += q[i]  
 y\_neLin\_result += y\_neLin[i]  
y\_for\_sigma=y\_result  
  
#Приведение к нужному виду  
y=y\_result\*\*2  
y\_neLin= y\_neLin\_result\*\*4  
q=q\_result  
y\_1 = integrate(y,(x,a,b))  
q\_1 = integrate(q,(x,a,b))  
y\_nelin\_1 = integrate(y\_neLin,(x,a,b))  
  
#Создание символов/Коэф.Интегралов  
Ee = Symbol('E')  
Hh = Symbol('h')  
Qq = Symbol('q')  
Ll = Symbol('l')  
y\_1\*=Ee\*(Hh\*\*3)/24  
q\_1\*=Qq  
y\_nelin\_1\*=(-1)\*(Ee\*(Hh\*\*5)\*Mm/120)  
  
#Массив производных  
Es=y\_1  
dw = []  
dw\_2 = []  
  
if(Choose == 0):  
 print("Q\_t =",Q\_\_T)  
 dw = Lin\_Function(w\_coefs,y\_1,q\_1,Q\_\_T)  
 dw\_2 = Ne\_Lin\_Function(w\_coefs, y\_1, q\_1, y\_nelin\_1,Q\_\_T)  
  
 print(Get\_W(L / 2, dw))  
 print(Get\_W(L / 2, dw\_2))  
 print("End")  
else:  
 fig, axs = plt.subplots(2)  
 dw= Ne\_Lin\_Function\_Loop(w\_coefs,y\_1,q\_1,y\_nelin\_1,fig, axs)  
 dw = Lin\_Function\_Loop(w\_coefs, y\_1, q\_1,fig, axs)  
  
  
  
#Данные для графика  
fig,axs = plt.subplots(2)  
x\_now\_1=0  
X\_count\_1 = L/0.5  
X\_step\_1 = L/X\_count\_1  
X\_for\_graph\_1 =[]  
W\_graph =[]  
  
# W для всех иксов  
if Choose == 0 :  
 for i in range (1,int(X\_count\_1)+2):  
 W\_result=0  
 W\_result = Get\_W(x\_now\_1,dw)  
 print(W\_result)  
 W\_graph.append(W\_result)  
 X\_for\_graph\_1.append(x\_now\_1)  
 x\_now\_1 +=X\_step\_1  
 #Вывод графика  
  
 axs[0].plot(X\_for\_graph\_1,W\_graph)  
  
  
 #Просчет сигм  
 y\_for\_sigma=y\_for\_sigma.subs([(Ll,L),(pi,m.pi)])  
 Sigma =[]  
 z=h/2  
 x\_now=0  
 X\_count = L/0.5  
 X\_step = L/X\_count  
 X\_for\_graph=[]  
 for i in range (1,int(X\_count)+2):  
 X\_for\_graph.append(x\_now)  
 E\_x =y\_for\_sigma.subs(x,x\_now)  
 x\_now+=X\_step  
 for j in range(1,N+1):  
 E\_x=E\_x.subs('w'+str(j),dw[j-1])  
  
 buf\_sigma = -1\*E\*z\*E\_x  
 Sigma.append(buf\_sigma)  
  
 #Вывод сигм  
 axs[1].plot(X\_for\_graph,Sigma)  
 plt.show()