Национальный исследовательский университет «Московский энергетический институт»

Курсовая работа

«Колебания тонкой пластины без учета потерь на трение»

Вариант 10.4

Выполнила

Самсонова Мария

Группа А-13а-19

Оглавление

[Постановка задачи 3](#_Toc91112815)

[Необходимый теоретический материал 4](#_Toc91112816)

[Построение тестового примера 7](#_Toc91112817)

[Численный метод 8](#_Toc91112818)

[Результаты расчетов по тестовым примерам 10](#_Toc91112819)

[Тестовый пример №1 10](#_Toc91112820)

[Тестовый пример №2 14](#_Toc91112821)

[Результаты вычислительного эксперимента 17](#_Toc91112822)

[Анализ полученных результатов 20](#_Toc91112823)

[Код с комментариями 21](#_Toc91112824)

[Использованная литература 25](#_Toc91112825)

# Постановка задачи

Нахождение колебаний тонкой пластины размером , где и , без учета потерь на трения, колебания которого выражаются волновым уравнением вида ,при граничных условиях , ,, (заданы в таблице) и начальных и

|  |  |  |
| --- | --- | --- |
|  |  | y  x  a  b |
|  |  |
|  |  |
|  |  |

# Необходимый теоретический материал

Воспользуемся теоретическим материалом для аналитического решения из книги А.Н.Тихонов, А.А.Самарский «УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ», а именно разделами

* Глава 2, §1 Поперечные колебания мембраны

Получено уравнение колебаний однородной мембраны дифференциальной форме

*,* где

* (\*)
* Глава 5, §3 Колебания прямоугольной мембраны
* Глава 3, §2 Метод разделения переменных

Основываясь на формуле (\*) построим общую задачу поиска формы мембраны без учета внешних сил

Решение найдем, используя *метод разделенных переменных.*

* Предположим (2)
* Подставив (2) в (1) получим

* Систему аналогично решим с помощью *метод разделенных переменных.*
* (4)
* Подставим (4) в (3) получим

(условия 2-го рода)

где

* Аналитически решая (5) и (6) получим

* Тогда

* Вернувшись к преобразованию (4) получим собственные функции
* Выберем так, чтобы норма функции была равна единице.
* Таким образом
* Подставив в получим
* В итоге получаем

*,* где

# Построение тестового примера

* + Предварительно отрезок переведем в отрезок и отрезок в отрезок
  + Чтобы построить тестовые примеры воспользуемся свойством собственных функций - Собственные функции образуют на [a,b] ортогональную систему:

Тестовый пример №1

Ожидаемый результат

Тестовый пример №2

Ожидаемый результат

# Численный метод

* Произведем дискретизацию задачи и применим аппроксимацию производных со вторым порядком точности в дифференциальном уравнении и с первым порядком точности в краевых условиях
* Преобразуем систему

Стратегия поиска

* Для k=0
* Для k=1

(1)

* + При j=0,M (Состояние пластины на границах и )
  + При i=0,N (Состояние пластины на границах и )
  + Значения неграничных точек найдем с помощью (1)
* Для значения будем искать по общей формуле

(2)

* + При j=0,M (Состояние пластины на границах и )
  + При i=0,N (Состояние пластины на границах и )
  + Значения неграничных точек найдем с помощью (2)

# Результаты расчетов по тестовым примерам

## Тестовый пример №1

Ожидаемый результат

В тестовом примере:

* Красный график – функция, построенная по разностной схеме
* Зеленый график – функция, построенная по аналитическому решению

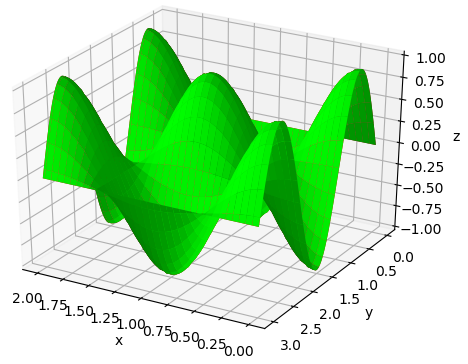
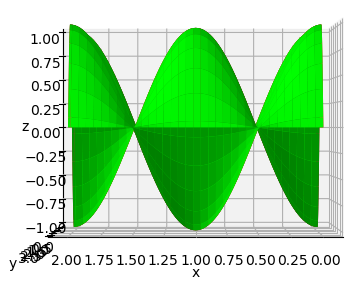
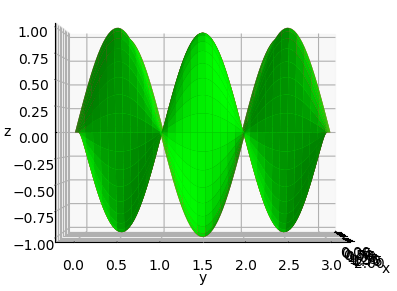
При этом использовано:

* 300 слоев (T = 3, dt = 0.01)
* шаг по оси X dx=0.02
* шаг по оси Y dy=0.03
* сохраняется сдвиг по двум осям: Ox , Oy

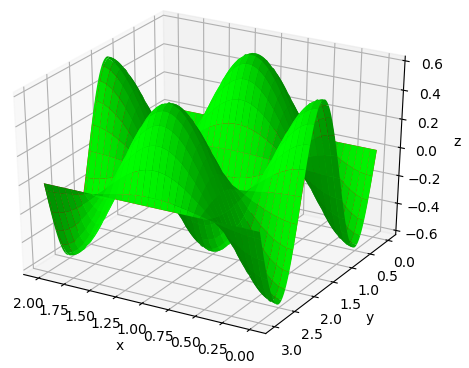
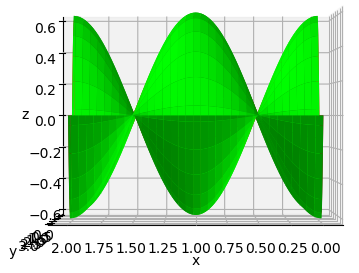
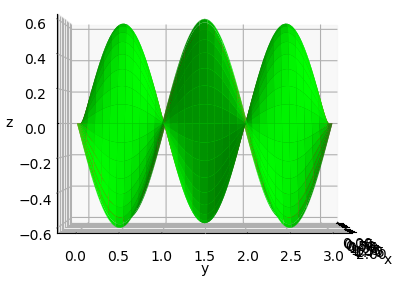
Результат в формате видео: [Тест1](https://youtu.be/Vvf_O0dKVJo)

Результаты по слоям:

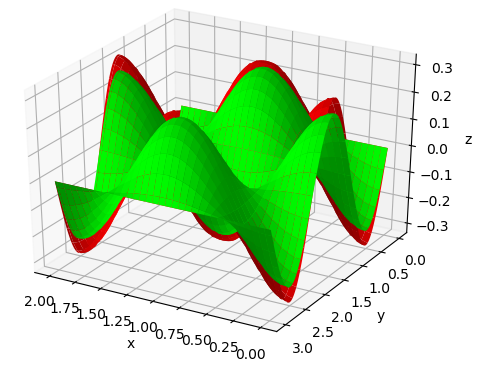
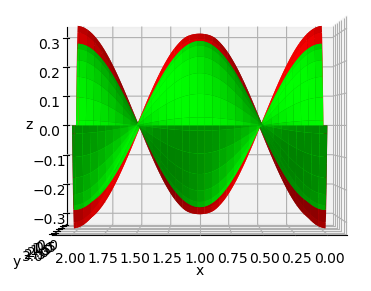
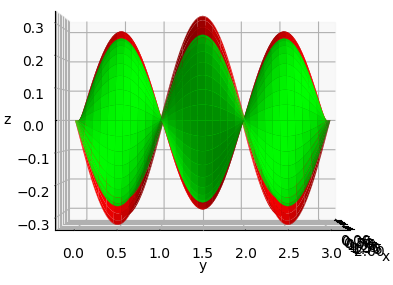
0 слой:



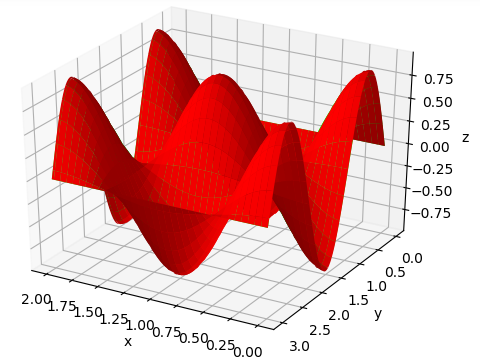
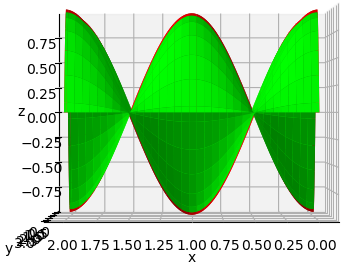
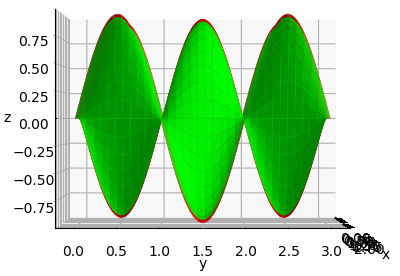
50 слой:



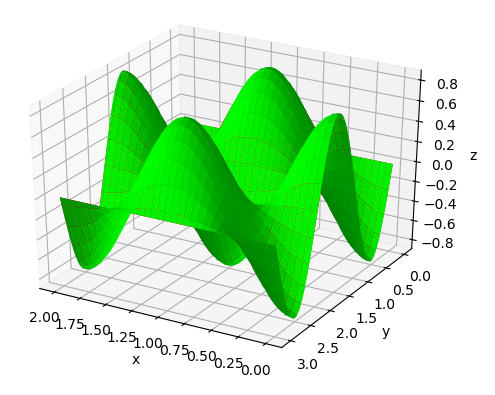
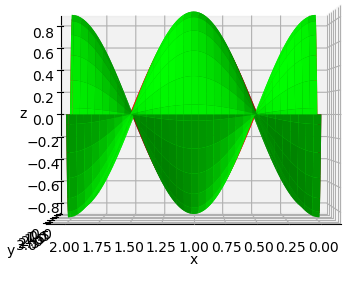
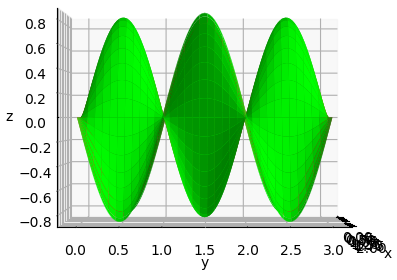
100 слой:



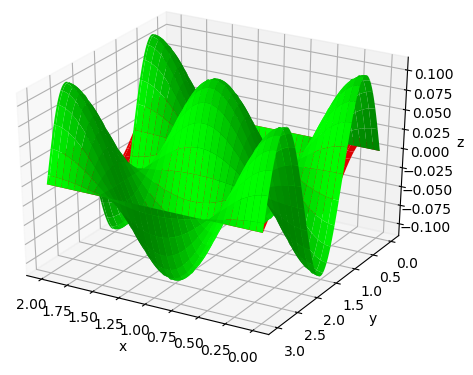
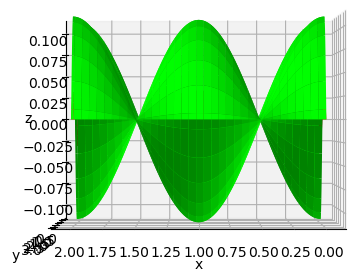
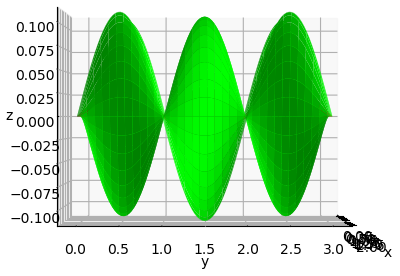
150 слой (пример впадины погрешности):

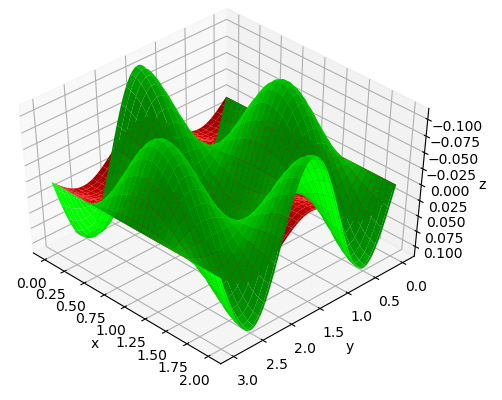


200 слой:

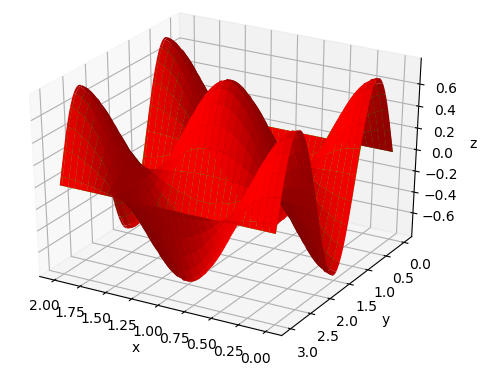
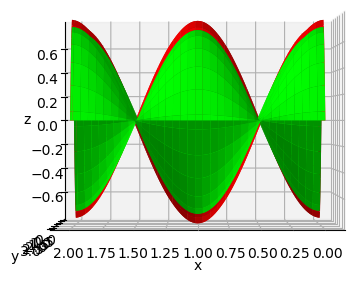
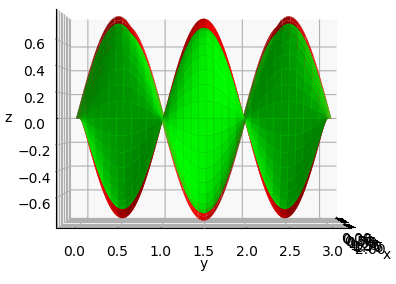


250 слой (пример пика погрешности):

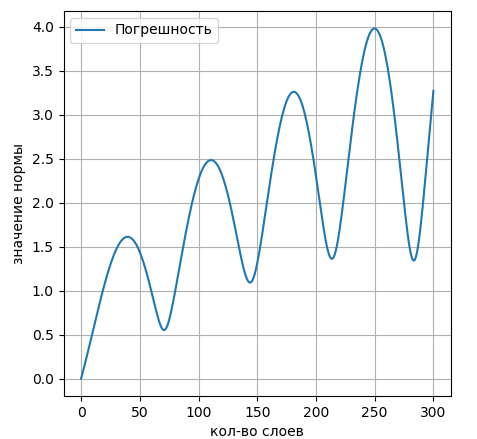




300 слой:



Определим поведение погрешности по слоям, для этого воспользуемся евклидовой нормой матрицы



Из графика видно, что с увеличением кол-ва слоев увеличивается и погрешность полученного слоя, причем пики графика погрешности соответствуют наименьшим (по модулю) значениям искомой функции (250 слой), и наоборот впадины – наибольшим (по модулю) значениям (150 слой).

Так же из графика погрешности можно примерно определить период функции: 0,7 секунд

## Тестовый пример №2

Ожидаемый результат

В тестовом примере:

* Красный график – функция, построенная по разностной схеме
* Зеленый график – функция, построенная по аналитическому решению

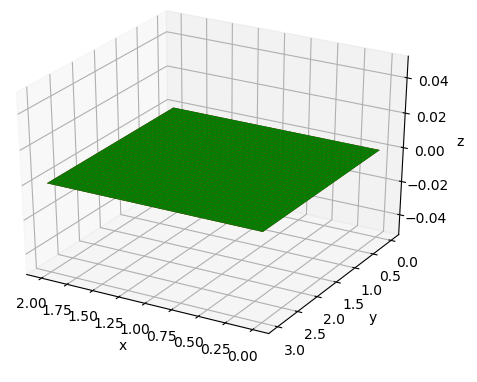
При этом использовано:

* 300 слоев (T = 3, dt = 0.01)
* шаг по оси X dx=0.02
* шаг по оси Y dy=0.03
* сохраняется сдвиг по двум осям: Ox , Oy

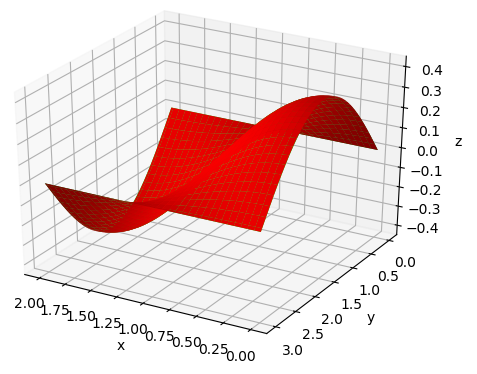
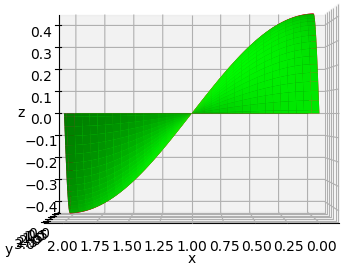
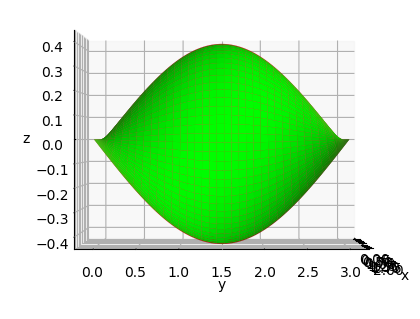
Результат в формате видео: [Тест2](https://youtu.be/gBfg68gk9-Y)

Результаты по слоям:

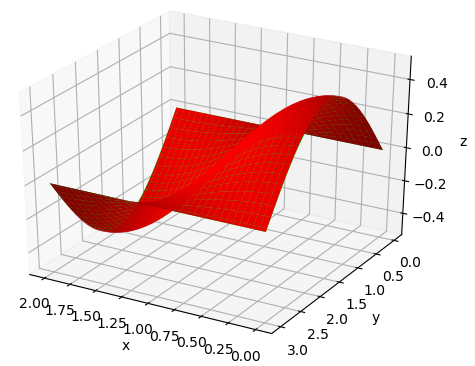
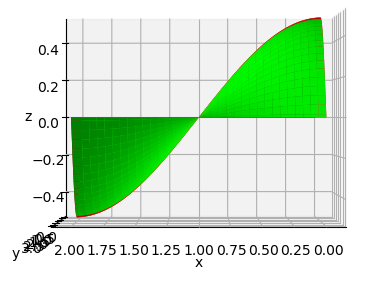
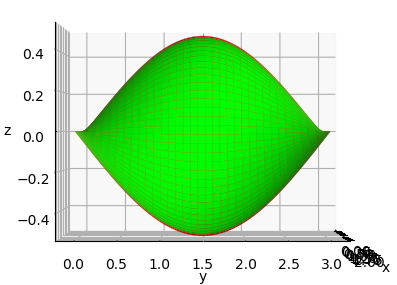
0 слой:



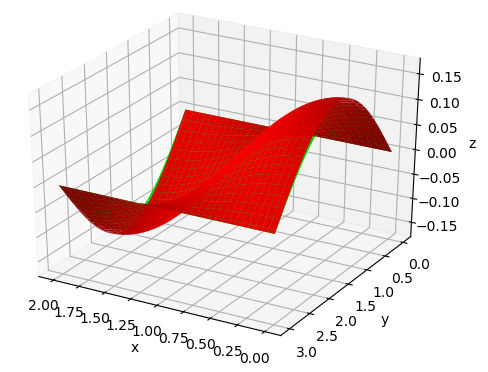
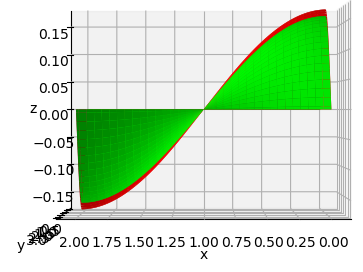
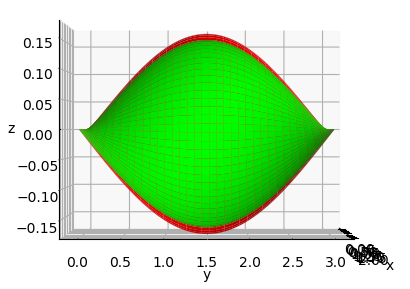
50 слой:



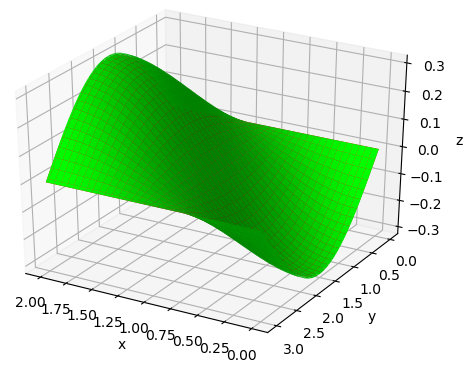
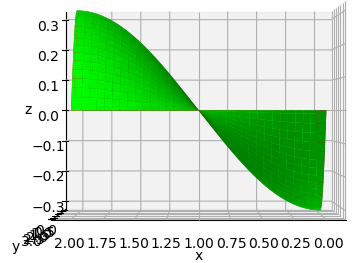
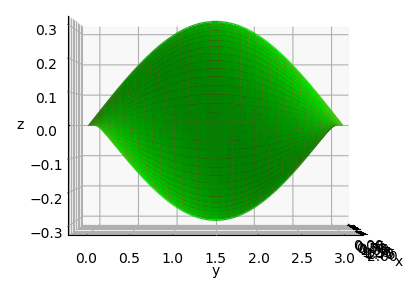
100 слой:



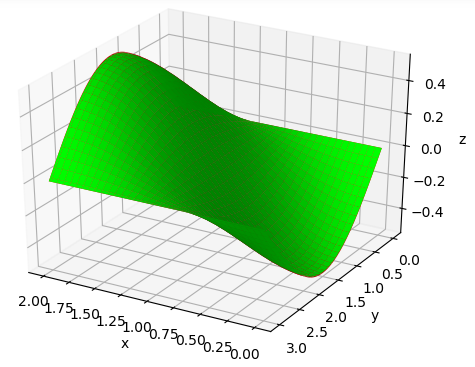
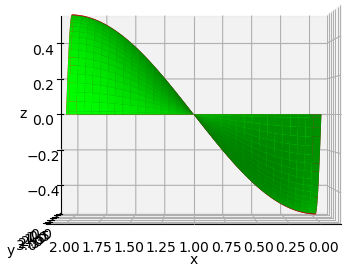
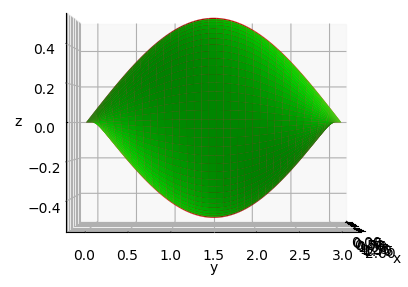
150 слой:



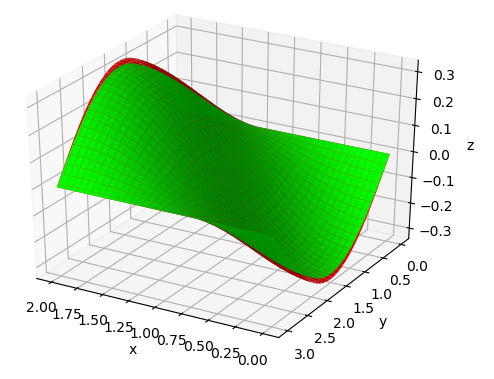
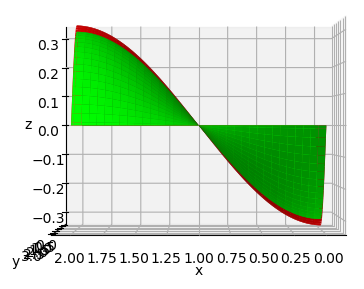
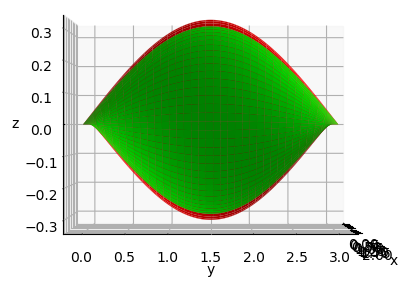
200 слой:



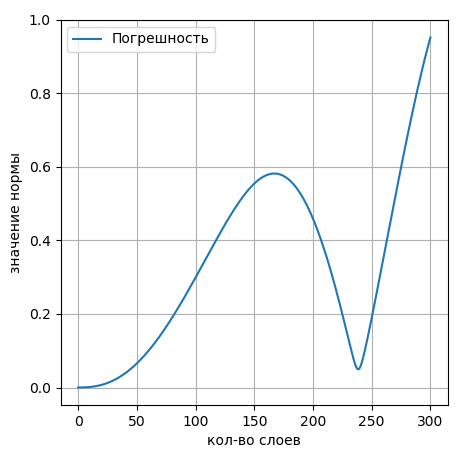
250 слой:



300 слой:



Аналогично прошлому примеру построим график погрешности



В отличие от прошлого примера, так как изменение конфигурации графика искомой функции происходят значительно медленнее, погрешность не имеет много пиков, однако сохраняется закономерность: погрешность выше там, где функция принимает наименьшие по модулю значения (150 слой, 300 слой) и ниже, где наибольшие по модулю значения (250 слой).

Так же из графика погрешности можно примерно определить период функции: 2,4 секунд

# Результаты вычислительного эксперимента

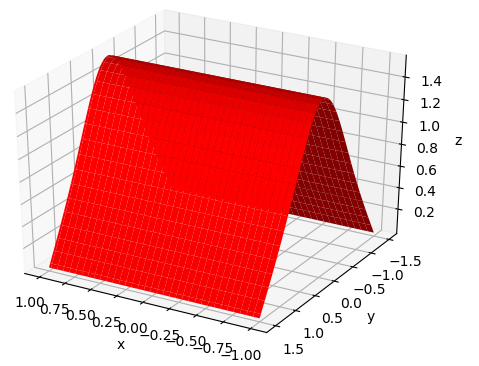
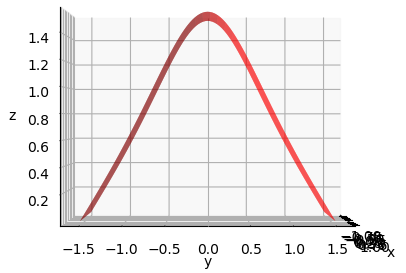
Функция построена при использовании:

* 600 слоев (T = 6, dt = 0.01)
* шаг по оси X dx=0.02
* шаг по оси Y dy=0.03
* сдвиг не используется: Ox , Oy

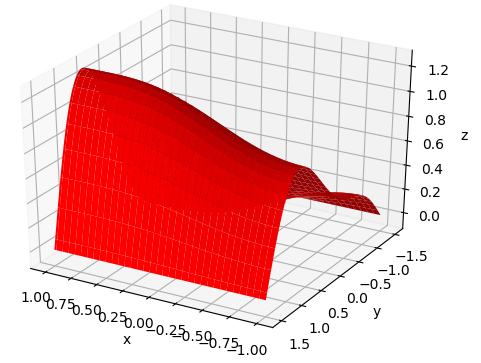
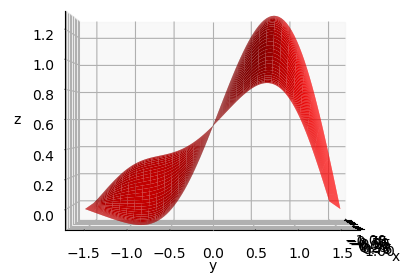
Результат в формате видео: [Решение](https://youtu.be/CIqS0LzF2mQ)

Результаты по слоям:

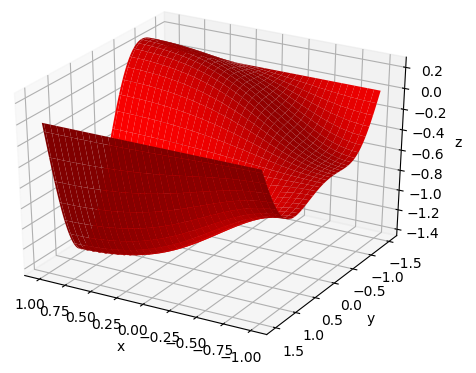
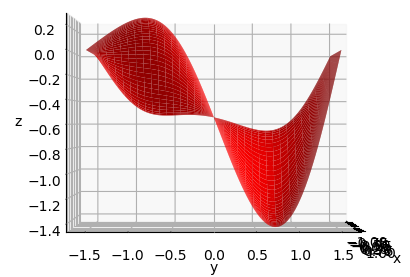
0 слой:



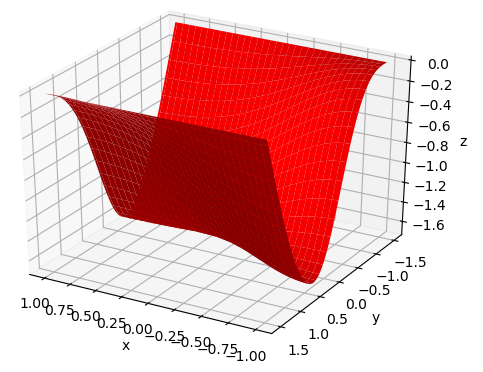
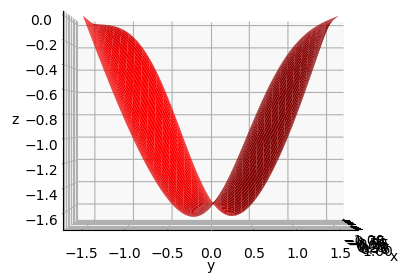
100 слой:



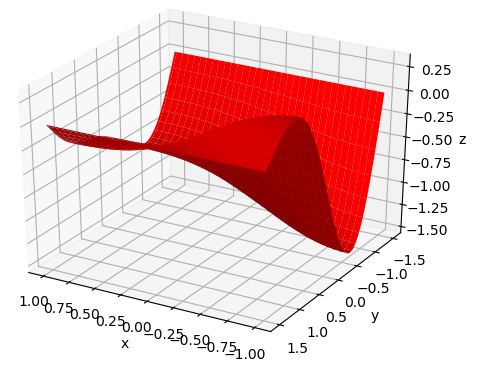
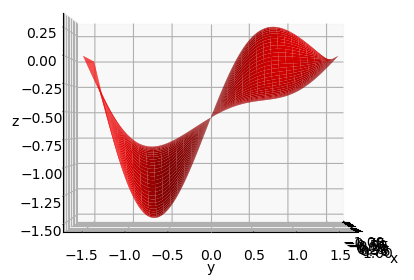
200 слой:



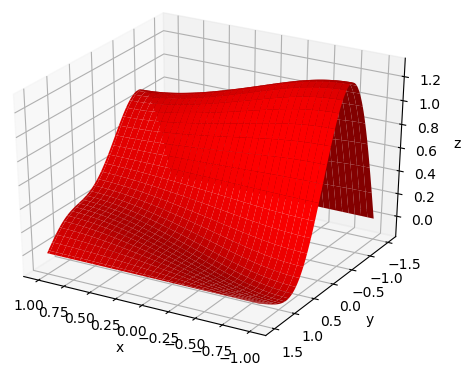
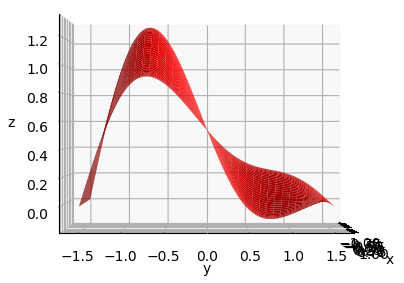
300 слой:



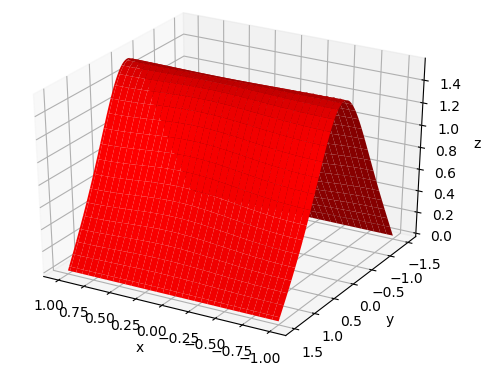
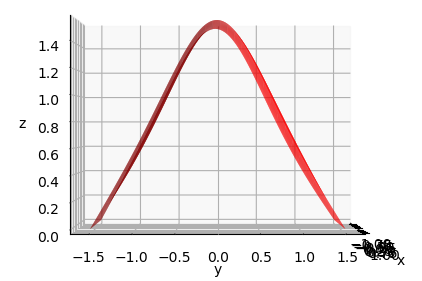
400 слой:



500 слой:



600 слой:



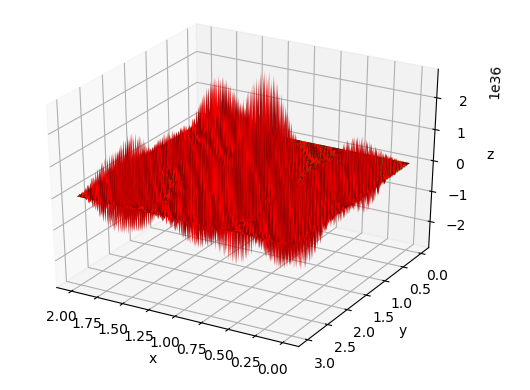
Из полученных графиков можно утверждать, что период приблизительно равен 6 секундам

# Анализ полученных результатов

Данная работа показала, что нахождение колебаний тонкой пластины с помощью разностной схемы 1 порядка дает хорошие результаты при малом количестве слоев. С увеличением количества слоев накапливается неточность и заметнее всего она проявляет себя на слоях с наименьшими по модулю значениями функции, что было показано на тестовых примерах. Так же при построении разностной схемы нужно грамотно подходить к выбору шагов по осям Ox, Oy и времени, так как при неудачном подборе наблюдается резкое возрастание значений функций, что приводит к переполнению типов данных. Например, при выборе для тестового примера №1:

* 100 слоев (T = 2, dt = 0.02)
* шаг по оси X dx=0.02
* шаг по оси Y dy=0.03

Получаем такой график на 100 слое:



Данную проблему можно избежать, используя соответствующие правила выбора шага или проверяя изменения слоев (не должно быть резких скачков).

# Код с комментариями

**import** matplotlib.pyplot **as** plt  
**from** mpl\_toolkits.mplot3d **import** Axes3D  
**import** math  
**import** numpy **as** np  
**from** celluloid **import** Camera  
  
*# Создание дискретных функций  
  
#Функция на нулевом слое***def** phi(n, a1, b1, i, j, dx, dy):  
 *# 1 тест* **if** (n == 1):  
 **return** math.cos(math.pi \* (a1 + i \* dx)) \* math.sin(math.pi \* (b1 + j \* dy))  
 *# 2 тест* **if** (n == 2):  
 **return** 0  
 *# решение задачи* **if** (n == 3):  
 **return** math.tan(math.cos((math.pi \* (b1 + j \* dy))/3))  
  
*#Производная функции на нулевом слое***def** psi(n, a1, b1, i, j, dx, dy):  
 *# 1 тест* **if** (n == 1):  
 **return** 0  
 *# 2 тест* **if** (n == 2):  
 **return** math.cos((math.pi \* (a1 + i \* dx)) / 2) \* math.sin((math.pi \* (b1 + j \* dy)) / 3)  
 *# решение задачи* **if** (n == 3):  
 **return** math.exp(math.sin((math.pi \* (a1 + i \* dx)) / 2)) \* math.sin((2 \* math.pi \* (b1 + j \* dy)) / 3)  
  
*#Функция на первом слое***def** omega(n, a1, b1, i, j, dx, dy, dt):  
 **return** psi(n, a1, b1, i, j, dx, dy) \* dt + phi(n, a1, b1, i, j, dx, dy)  
  
*#0 слой***def** Layer0(n, a1, b1, nx, ny, dx, dy, dt):  
 u0 = np.zeros((nx, ny))  
 **for** i **in** range(nx):  
 **for** j **in** range(ny):  
 u0[i][j] = phi(n, a1, b1, i, j, dx, dy)  
  
 **return** u0  
  
*#1 слой***def** Layer1(n, a1, b1, nx, ny, dx, dy, dt):  
 u1 = np.zeros((nx, ny))  
 **for** i **in** range(nx):  
 **for** j **in** range(ny):  
 u1[i][j] = omega(n, a1, b1, i, j, dx, dy, dt)  
  
 **return** u1  
  
*#2 слой***def** Layer2(n, a1, b1, nx, ny, dx, dy, dt):  
 u2 = np.zeros((nx, ny))  
 **for** i **in** range(nx):  
 **for** j **in** range(ny):  
 **if** (j == 0 **or** j == (ny - 1)):  
 u2[i][j] = 0  
 **elif** (i == 0):  
 u2[i][j] = 2 \* omega(n, a1, b1, i, j, dx, dy, dt) - phi(n, a1, b1, i, j, dx, dy) + ((dt \* dt) / (dx \* dx)) \* (  
 -omega(n, a1, b1, i, j, dx, dy, dt) + omega(n, a1, b1, i + 1, j, dx, dy, dt)) + ((dt \* dt) / (dy \* dy)) \* (  
 omega(n, a1, b1, i, j - 1, dx, dy, dt) - 2 \* omega(n, a1, b1, i, j, dx, dy, dt) + omega(n,a1, b1, i, j + 1, dx, dy,dt))  
 **elif** (i == (nx - 1)):  
 u2[i][j] = 2 \* omega(n, a1, b1, i, j, dx, dy, dt) - phi(n, a1, b1, i, j, dx, dy) + ((dt \* dt) / (dx \* dx)) \* (  
 -omega(n, a1, b1, i, j, dx, dy, dt) + omega(n, a1, b1, i - 1, j, dx, dy, dt)) + ((dt \* dt) / (dy \* dy)) \* (  
 omega(n, a1, b1, i, j - 1, dx, dy, dt) - 2 \* omega(n, a1, b1, i, j, dx, dy, dt) + omega(n, a1, b1, i, j + 1, dx, dy,dt))  
 **else**:  
 u2[i][j] = 2 \* omega(n, a1, b1, i, j, dx, dy, dt) - phi(n, a1, b1, i, j, dx, dy) + ((dt \* dt) / (dx \* dx)) \* (  
 omega(n, a1, b1, i - 1, j, dx, dy, dt) - 2 \* omega(n, a1, b1, i, j, dx, dy, dt) + omega(n, a1, b1, i + 1, j, dx, dy,dt)) + (  
 (dt \* dt) / (dy \* dy)) \* (omega(n, a1, b1, i, j - 1, dx, dy, dt) - 2 \* omega(n, a1, b1, i, j, dx, dy, dt) + omega(n, a1, b1, i, j + 1,dx, dy, dt))  
  
 **return** u2  
  
*#3+ слой***def** Layer(k, u, nx, ny, dx, dy, dt):  
 u3 = np.zeros((nx, ny))  
 **for** i **in** range(nx):  
 **for** j **in** range(ny):  
 **if** (j == 0 **or** j == (ny - 1)):  
 u3[i][j] = 0  
 **elif** (i == 0):  
 u3[i][j] = 2 \* u[k][i][j] - u[k - 1][i][j] + ((dt \* dt) / (dx \* dx)) \* (  
 -u[k][i][j] + u[k][i + 1][j]) + ((dt \* dt) / (dy \* dy)) \* (  
 u[k][i][j - 1] - 2 \* u[k][i][j] + u[k][i][j + 1])  
 **elif** (i == (nx - 1)):  
 u3[i][j] = 2 \* u[k][i][j] - u[k - 1][i][j] + ((dt \* dt) / (dx \* dx)) \* (  
 -u[k][i][j] + u[k][i - 1][j]) + ((dt \* dt) / (dy \* dy)) \* (  
 u[k][i][j - 1] - 2 \* u[k][i][j] + u[k][i][j + 1])  
 **else**:  
 u3[i][j] = 2 \* u[k][i][j] - u[k - 1][i][j] + ((dt \* dt) / (dx \* dx)) \* (  
 u[k][i - 1][j] - 2 \* u[k][i][j] + u[k][i + 1][j]) + ((dt \* dt) / (dy \* dy)) \* (  
 u[k][i][j - 1] - 2 \* u[k][i][j] + u[k][i][j + 1])  
 **return** u3  
  
*# Заполнение слоев***def** Layers(n, a1, a2, b1, b2, T, dx, dy, dt):  
 nx = int((a2 - a1) / dx) + 1  
 ny = int((b2 - b1) / dy) + 1  
 nt = int(T / dt) + 1  
  
 u = np.zeros((nt, nx, ny))  
  
 **for** k **in** range(nt):  
 **if** (k == 0):  
 u[k] = Layer0(n, a1, b1, nx, ny, dx, dy, dt)  
 **elif** (k == 1):  
 u[k] = Layer1(n, a1, b1, nx, ny, dx, dy, dt)  
 **elif** (k == 2):  
 u[k] = Layer2(n, a1, b1, nx, ny, dx, dy, dt)  
 **else**:  
 u[k] = Layer(k - 1, u, nx, ny, dx, dy, dt)  
  
 **return** u  
  
*#Функция полученная аналитически (для тестовых примеров)***def** UTest(n, a1, a2, b1, b2, T, dx, dy, dt):  
 nx = int((a2 - a1) / dx) + 1  
 ny = int((b2 - b1) / dy) + 1  
 nt = int(T / dt) + 1  
 ut = np.zeros((nt, nx, ny))  
  
 *# 1 тест* **if** (n == 1):  
 **for** k **in** range(nt):  
 **for** i **in** range(nx):  
 **for** j **in** range(ny):  
 ut[k][i][j] = math.cos((a1 + i \* dx) \* math.pi) \* math.sin((b1 + j \* dy) \* math.pi) \* math.cos((2 \*\* 0.5) \* math.pi \* (k \* dt))  
 *# 2 тест* **if** (n == 2):  
 **for** k **in** range(nt):  
 **for** i **in** range(nx):  
 **for** j **in** range(ny):  
 ut[k][i][j] = (6 / ((13 \*\* 0.5)\* math.pi))\* math.cos((a1 +i \* dx) \* (math.pi/2)) \* math.sin((b1 + j \* dy) \* (math.pi/3)) \* math.sin(((13 \*\* 0.5) \* math.pi \* (k \* dt))/6)  
 **return** ut  
  
*#значения длин сторон*a=2  
b=3  
  
*#шаги разностной схемы*dx = 0.02  
dy = 0.03  
dt = 0.01  
  
*#общее время*T = 6  
  
*#кол-во разбиений отрезка*nx = 100 *#по оси Ox и Oy*nt = 600 *#по оси времени (кол-во слоев)  
  
#номер задачи (1 - тест 1, 2 - тест 2, 3 - поставленная задача)*n = 3  
  
**if** (n == 1):  
 *# 1 тестовый пример* ut = UTest(n, 0, a, 0, b, T, dx, dy, dt)  
 u = Layers(n, 0, a, 0, b, T, dx, dy, dt)  
 x = np.linspace(0, a, nx + 1)  
 y = np.linspace(0, b, nx + 1)  
 namefile = **'animation\_test1.gif'  
elif** (n == 2):  
 *# 2 тестовый пример* ut = UTest(n, 0, a, 0, b, T, dx, dy, dt)  
 u = Layers(n, 0, a, 0, b, T, dx, dy, dt)  
 x = np.linspace(0, a, nx + 1)  
 y = np.linspace(0, b, nx + 1)  
 namefile = **'animation\_test2.gif'  
else**:  
 *# решение поставленной задачи* u = Layers(3, -(a / 2), (a / 2), -(b / 2), (b / 2), T, dx, dy, dt)  
 x = np.linspace(-(a / 2), (a / 2), nx + 1)  
 y = np.linspace(-(b / 2), (b / 2), nx + 1)  
 namefile = **'animation.gif'***# Создание анимации с помощью библиотек matplotlib и celluloid*fig = plt.figure()  
ax = fig.add\_subplot(111, projection=**'3d'**)  
camera = Camera(fig)  
x, y = np.meshgrid(x, y, indexing = **'ij'**)  
  
**for** i **in** range(int(nt/10)+1):  
 U=ax.plot\_surface(x, y, u[i\*10], color = (1, 0, 0))  
  
 **if**(n != 3):  
 UT=ax.plot\_surface(x, y, ut[i\*10], color = (0, 1, 0))  
  
 ax.set(xlabel=**'x'**, ylabel=**'y'**, zlabel=**'z'**)  
 camera.snap()  
  
animation = camera.animate(interval = 300, repeat = **True**, repeat\_delay = 500)  
animation.save(namefile, writer = **'imagemagick'**, dpi=100)  
  
*#функция нахождения норм матриц погрешностей на всех слоях***def** Err(u, ut, a1, a2, b1, b2, T, dx, dy, dt):  
 nx = int((a2 - a1) / dx) + 1  
 ny = int((b2 - b1) / dy) + 1  
 nt = int(T / dt) + 1  
  
 errmat = np.zeros((nt, nx, ny))  
 normmat = np.zeros(nt)  
  
 **for** k **in** range(nt):  
 **for** i **in** range(nx):  
 **for** j **in** range(ny):  
 errmat[k][i][j] = abs(u[k][i][j] - ut[k][i][j])  
  
 normmat[k] = np.linalg.norm(errmat[k], ord=2)  
  
 **return** normmat  
  
*#массив норм*normmat = Err(u, ut, 0, a, 0, b, T, dx, dy, dt)  
  
*#вывод графика погрешности*plt.figure(figsize=(5, 5))  
plt.xlabel(**"кол-во слоев"**) *# ось абсцисс*plt.ylabel(**"значение нормы"**) *# ось ординат*plt.grid() *# включение отображение сетки*x = np.linspace(0, nt, (int(T / dt) + 1))  
y = normmat  
  
plt.plot(x, y, label=**'Погрешность'**)  
  
plt.legend() *# легенда*

# Использованная литература

* А.Н.Тихонов, А.А.Самарский «УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ»
* А.А.Амосов, Ю.А. Дубинский, Н.В.Копченова «ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ»