|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**Лабораторная работа №4**

**«Применение базовых методов решения ДУЧП2 параболического типа»**

**ДИСЦИПЛИНА: «Моделирование»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-62Б | |  |  | ( | Калашников А.С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Никитенко У.В. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

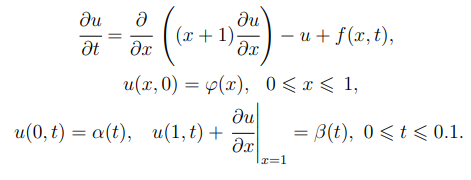
|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

Калуга, 2023

**Цель** **работы**: сформировать практические навыки анализа возможностей построения и выделения наиболее важных свойств объектов моделей для моделирования и использования специализированных программных пакетов и библиотек для стандартных вычислений и визуализации результатов численного или приближенно-аналитического решения ДУЧП2 параболического типа на основе сравнения результатов.

**Задачи**: решить уравнение, указанное в варианте методом аппроксимации дифференциального оператора. Выбрать среду для проведения расчетов и вычислительного эксперимента. Написать программу, реализующую решение задачи. Оценить результаты расчетов. Визуализировать результаты.

**Вариант №8**

****

Используя различные разностные схемы

* явную схему порядка с аппроксимацией производных в граничных условиях с порядком ;
* схему с весами при σ = 0, σ = 1, σ = 1/2 с аппроксимацией производных в граничных условиях с порядком .

По решению задачи должен быть представлен отчет, содержащий

1) Алгоритм решения задачи.

2) Тестирование алгоритма на решениях, для которых разностная схема точно аппроксимирует дифференциальную задачу.

3) Тестирование алгоритма, например, на решениях , на которых разностная схема неточно аппроксимирует дифференциальную задачу.

4) Таблицы решения на «крупной» сетке независимо от шагов по t и x, с которыми строится решение, следующего вида ().

5) Таблицы, характеризующие точность решения и внутреннюю сходимость.

**Ход решения с явной разностной схемой**

Алгоритм решения

Найдем Lu:

Найдем :

Найдем значение :

Находим

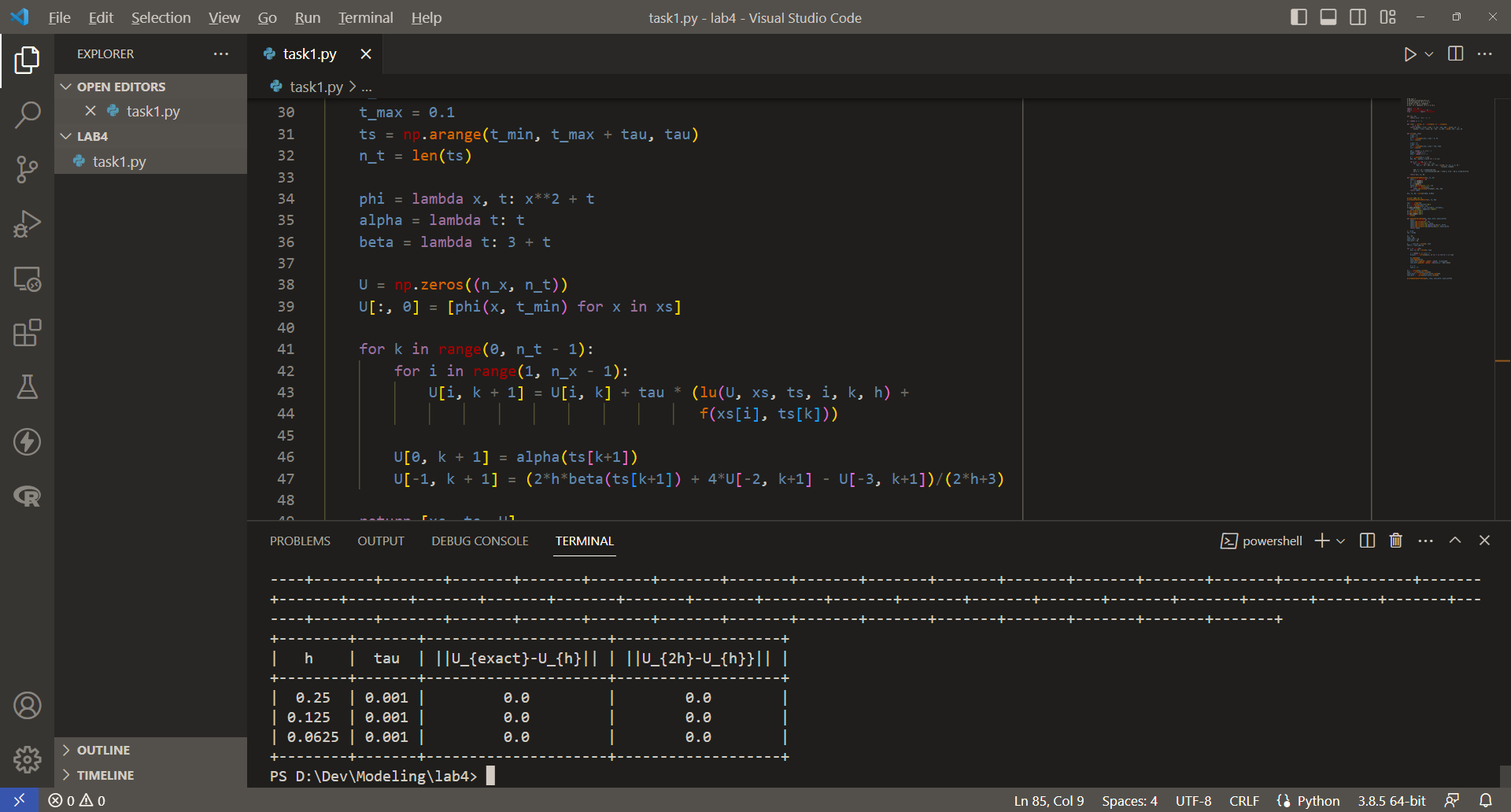
Вычислим граничные условия:

Запишем исходное выражение с использованием узловых значений:

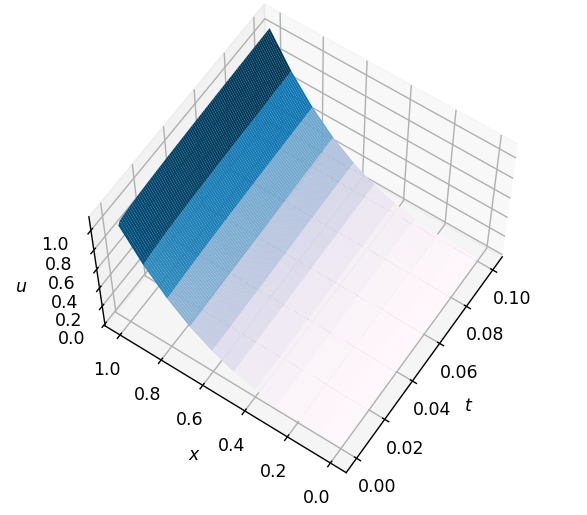
Выразим текущее значение из предыдущих:

Для проверки точного решения используем функцию , тогда

При подстановке функций в программу получим 0 погрешность

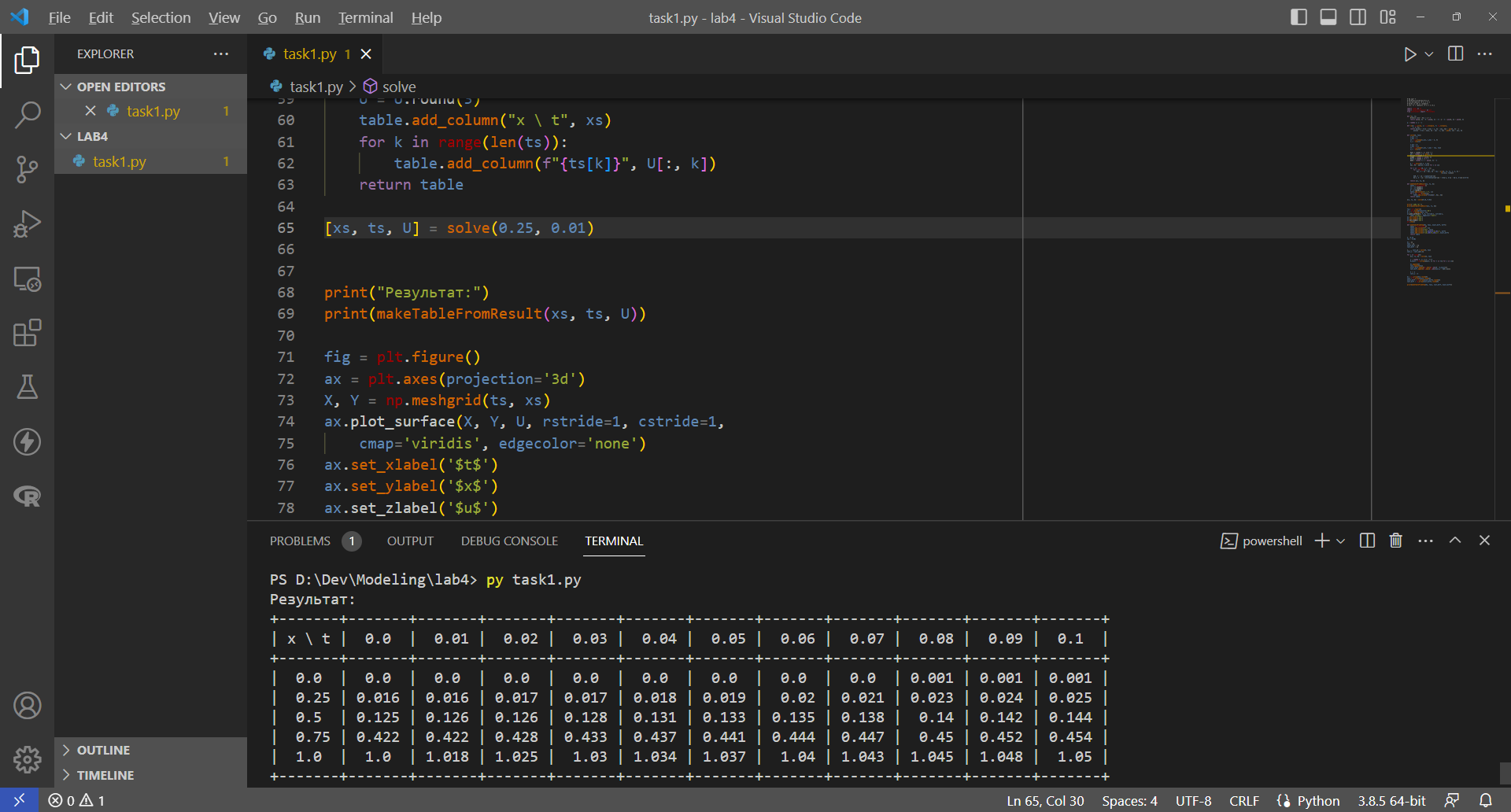


**Рисунок 1** Результат работы явной схемы для точной аппроксимации

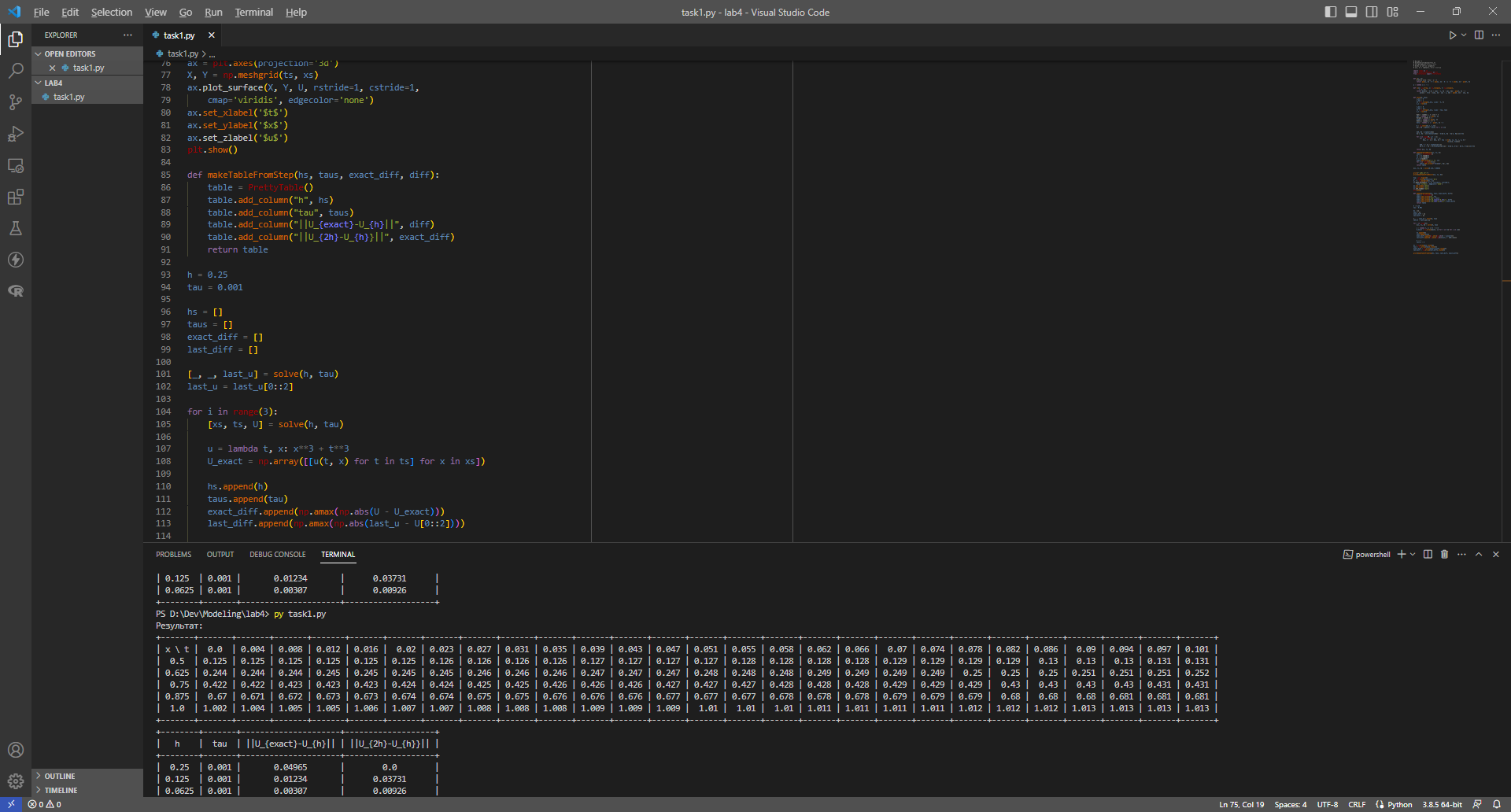


**Рисунок 2** График аппроксимированной функции

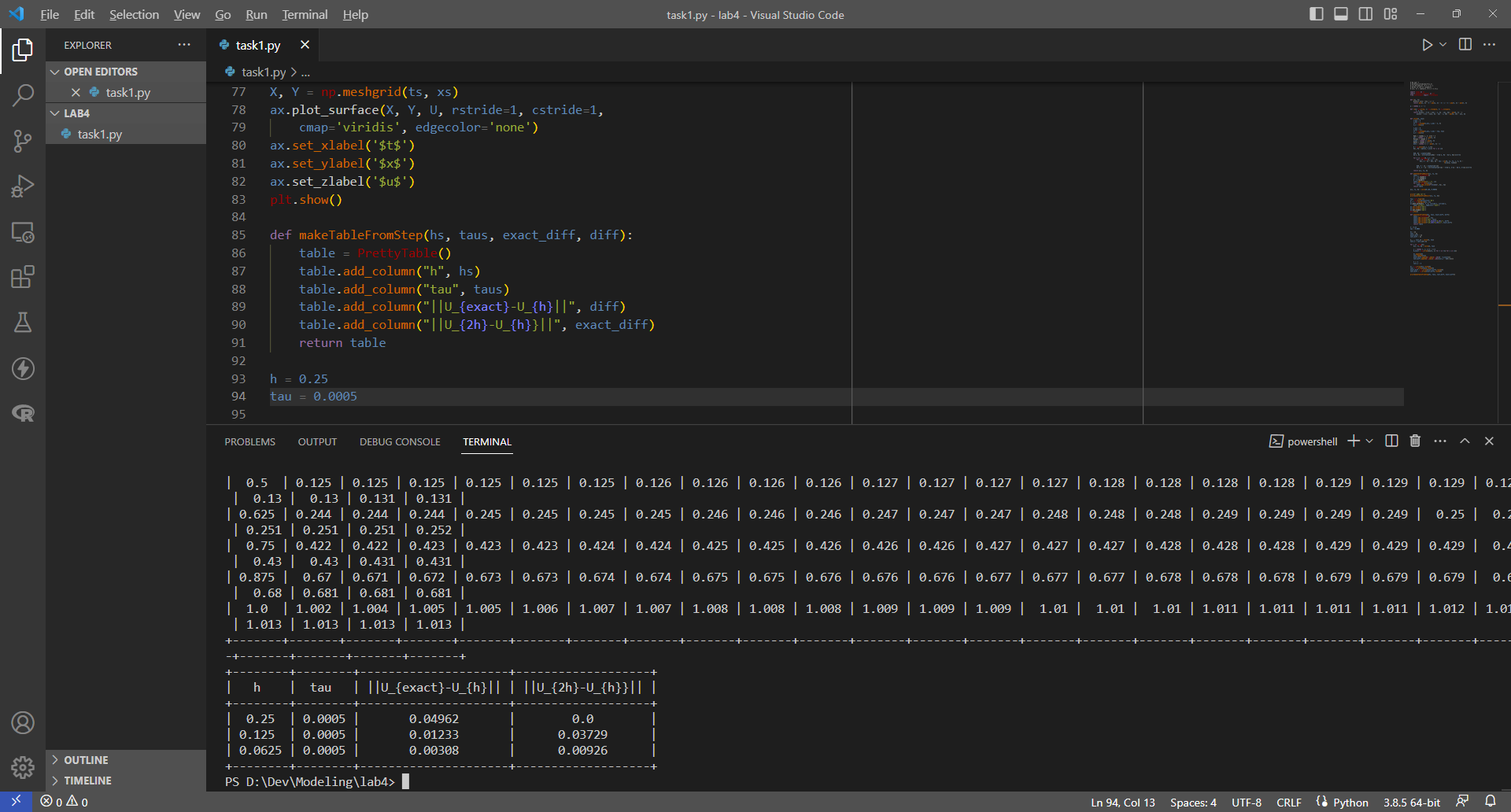
Протестируем алгоритм разностной схемы на функции , тогда



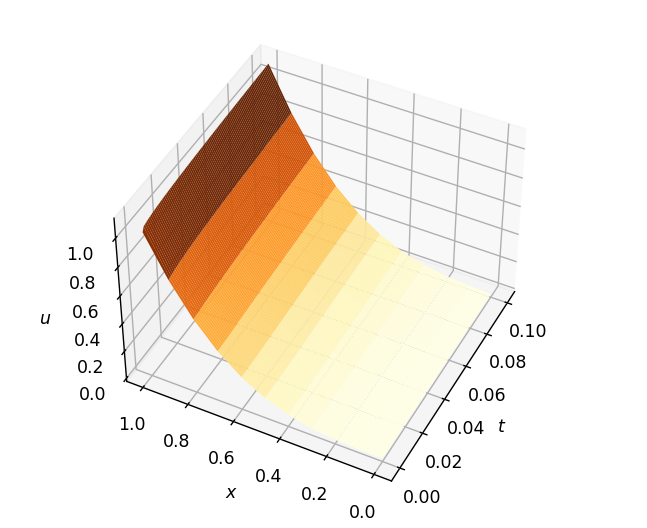
**Рисунок 3** Результат аппроксимации при



**Рисунок 4** Результат аппроксимации при



**Рисунок 5**Таблица, характеризующая точность полученного решения



**Рисунок 6** График аппроксимированной функции

Таким образом, с уменьшением шага по при фиксированном шаге по растёт точность аппроксимации.

**Ход решения с использованием схемы с весами**

Алгоритм решения

Найдем Lu:

Найдем :

Найдем значение :

Найдём начальные условия:

Вычислим граничные условия:

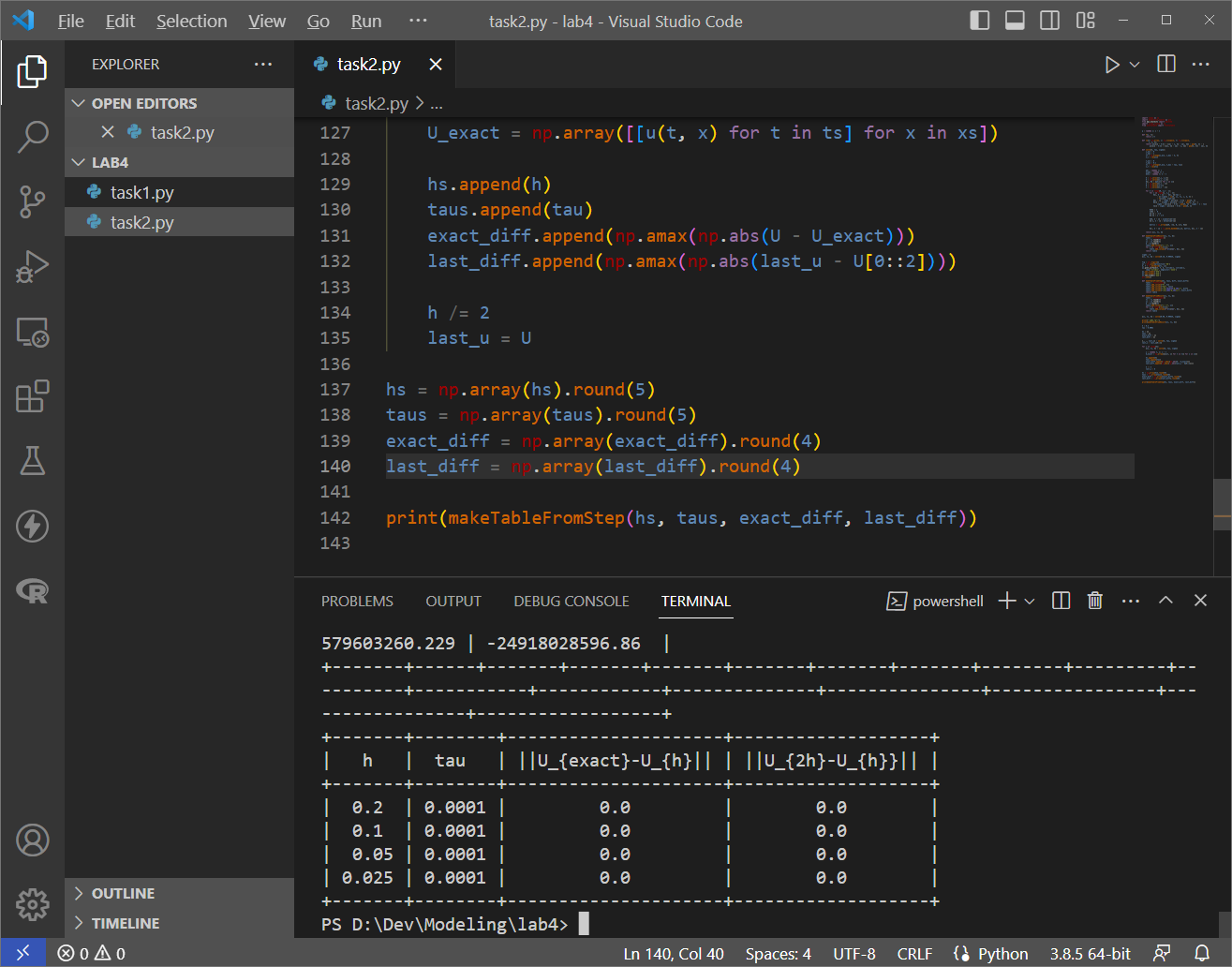
Для решения на каждом последующем слое необходимо решить систему уравнений. Составим коэффициенты для этой системы.

Подставим значение из прошлого случая:

Исходя из полученного уравнения и граничных условий, можно составить следующую систему:

Для проверки точного решения возьмём функцию , тогда

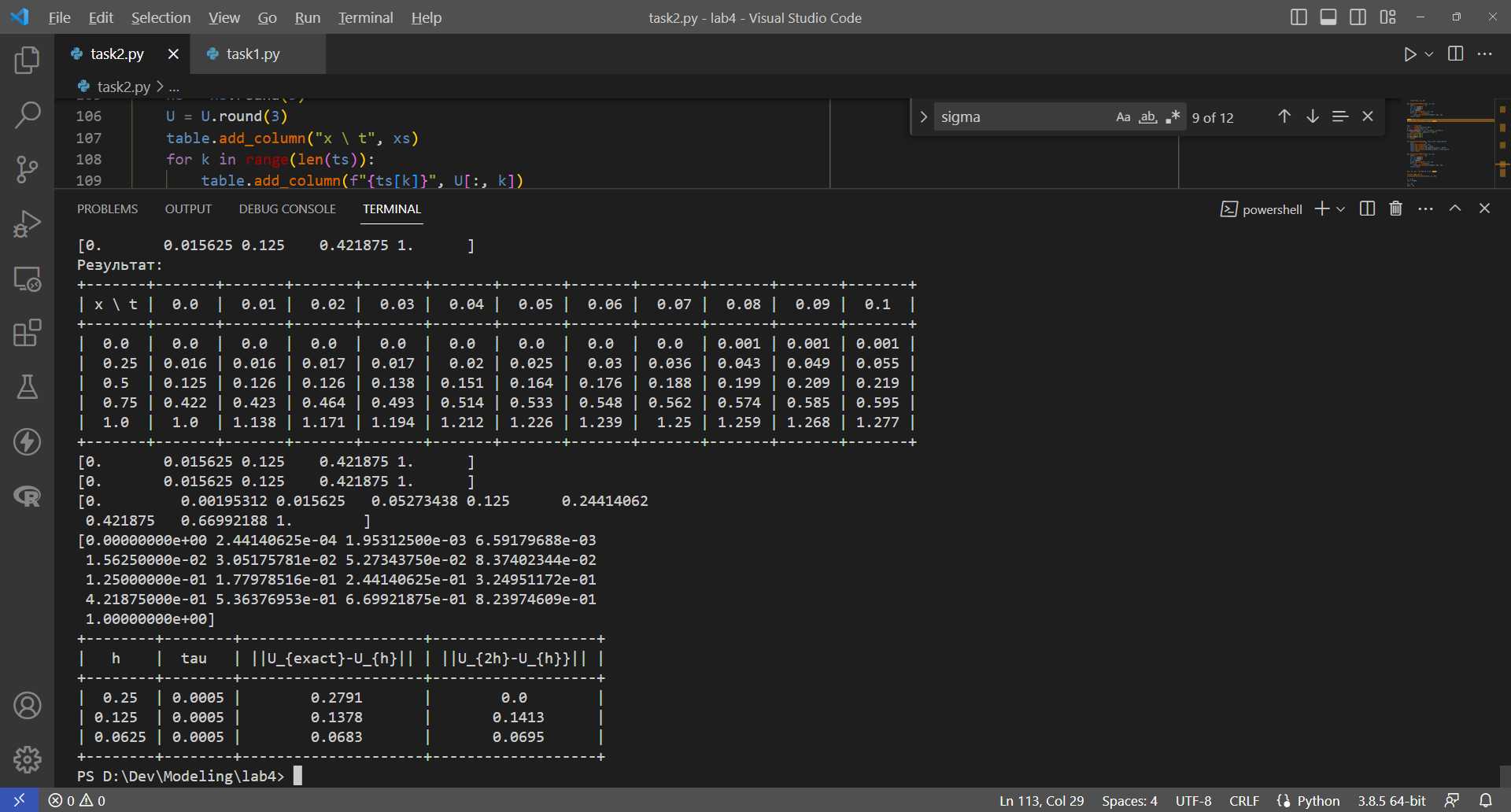
Проверим правильность для точной аппроксимации при   
.



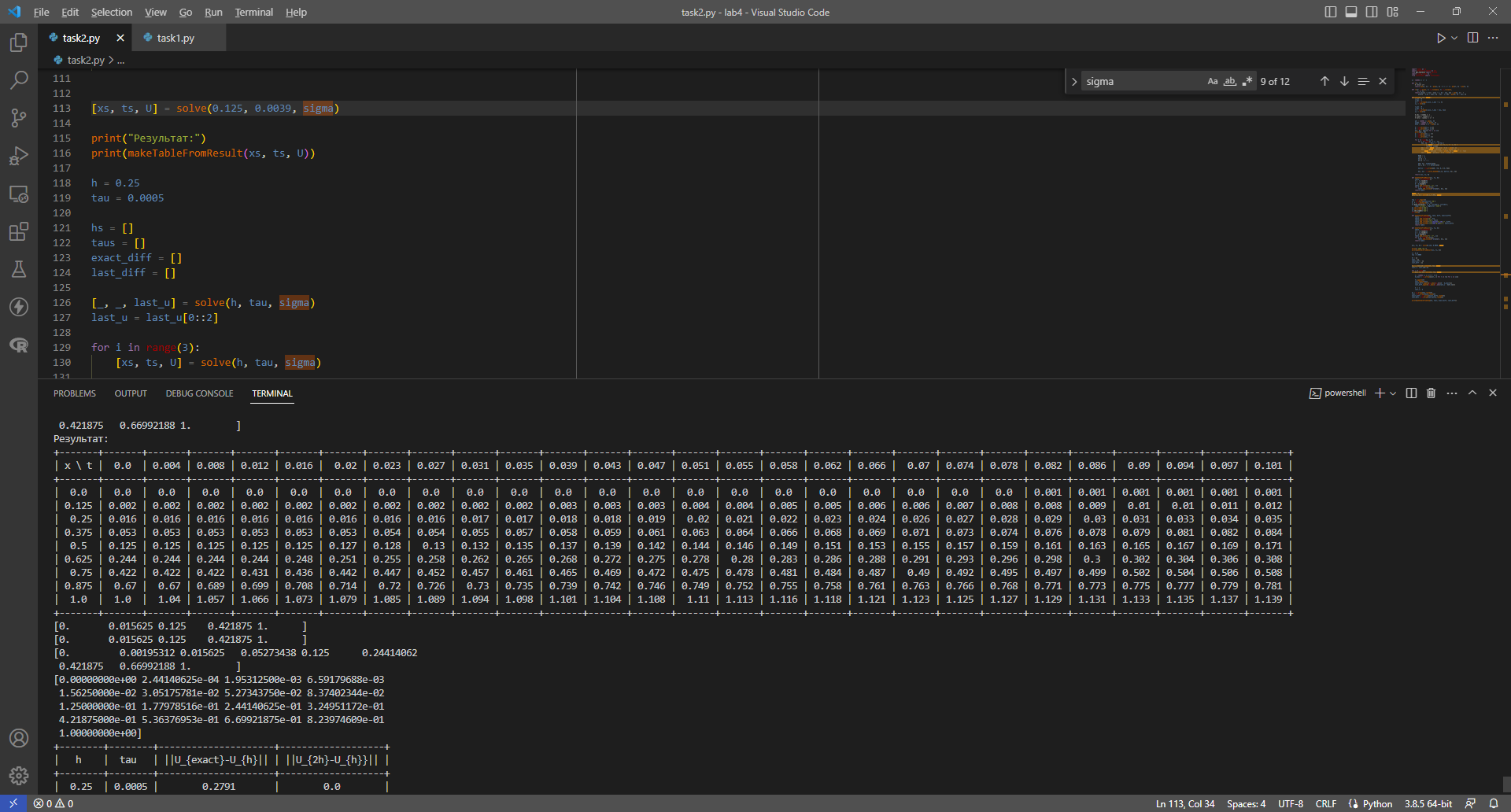
**Рисунок 7** Таблица, характеризующая точность решения для точной функции

Аппроксимируем функцию :

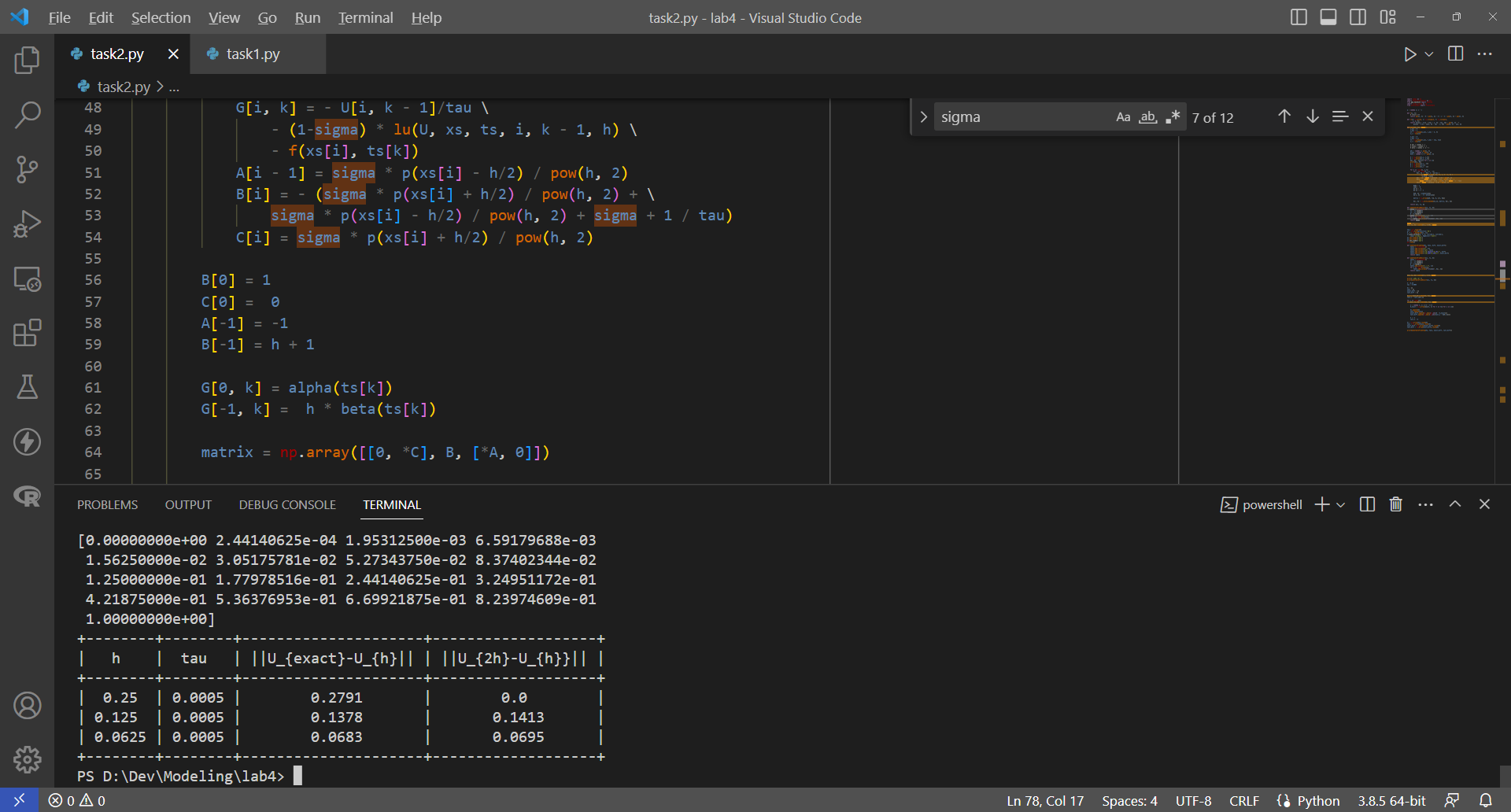
Вычислим значения для функции при .



**Рисунок 8** Результат аппроксимации при

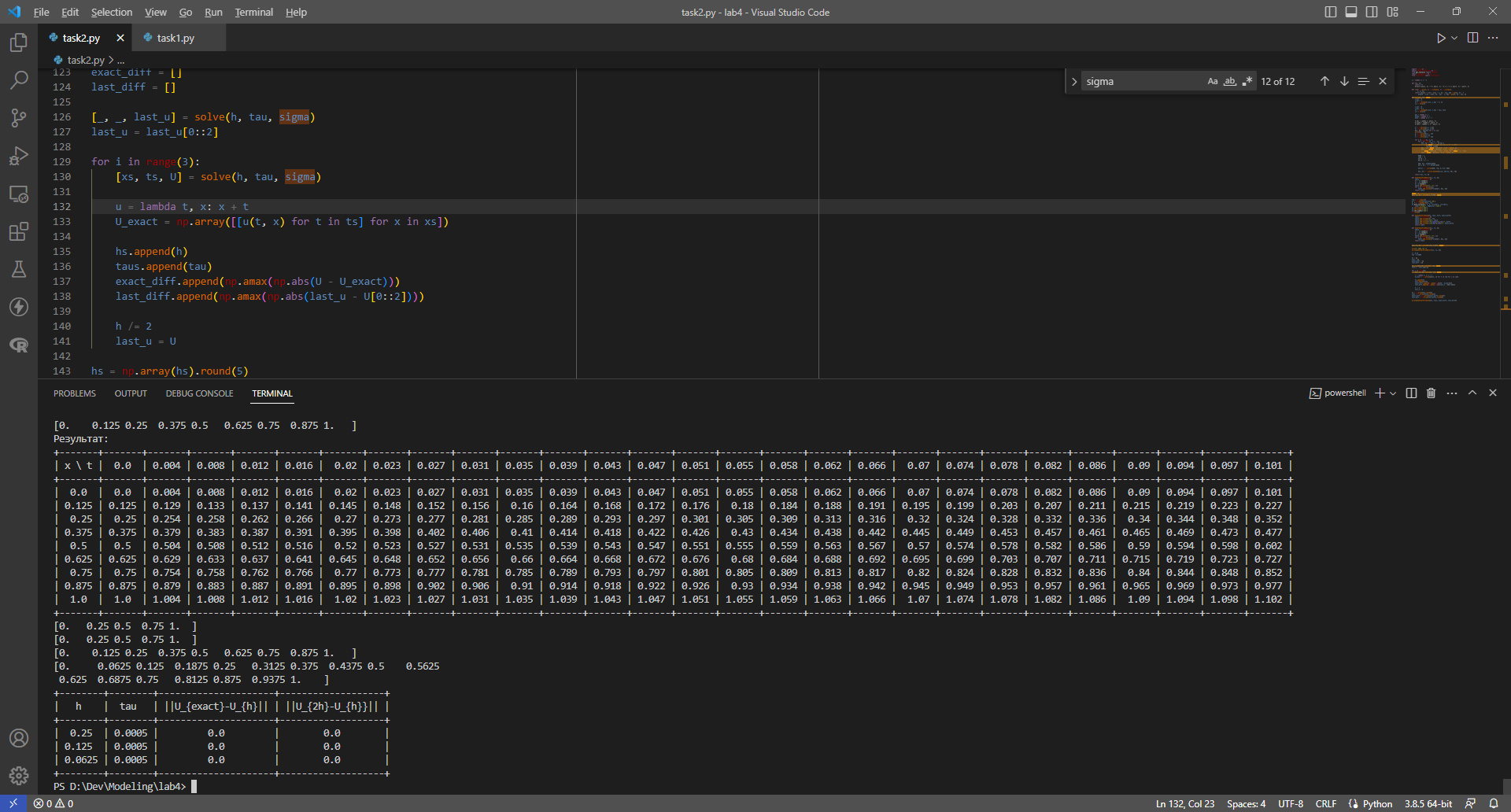


**Рисунок 9** Результат аппроксимации при



**Рисунок 11** Таблица, характеризующая точность полученного решения

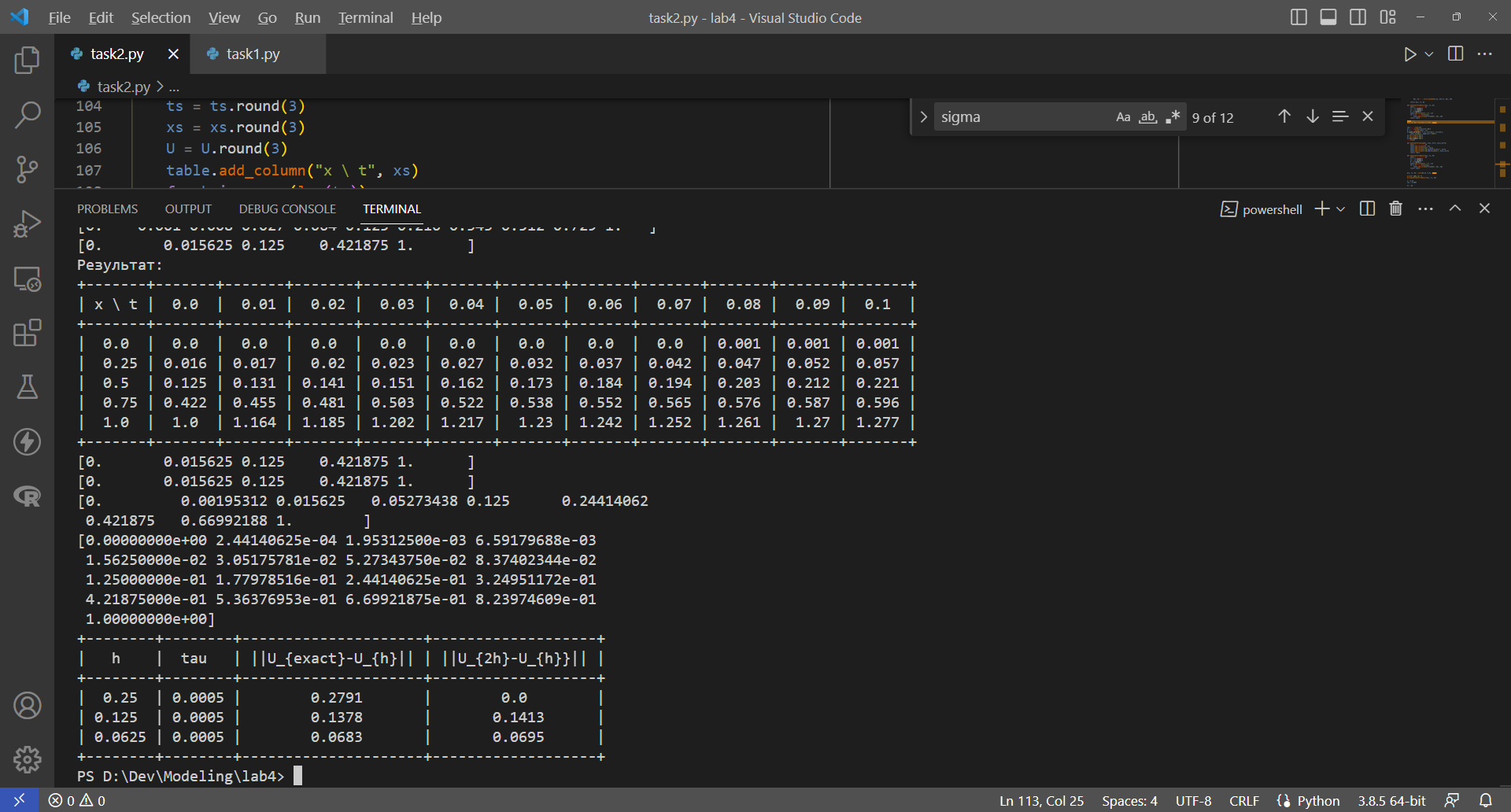
Проверим правильность для точной аппроксимации при   
.



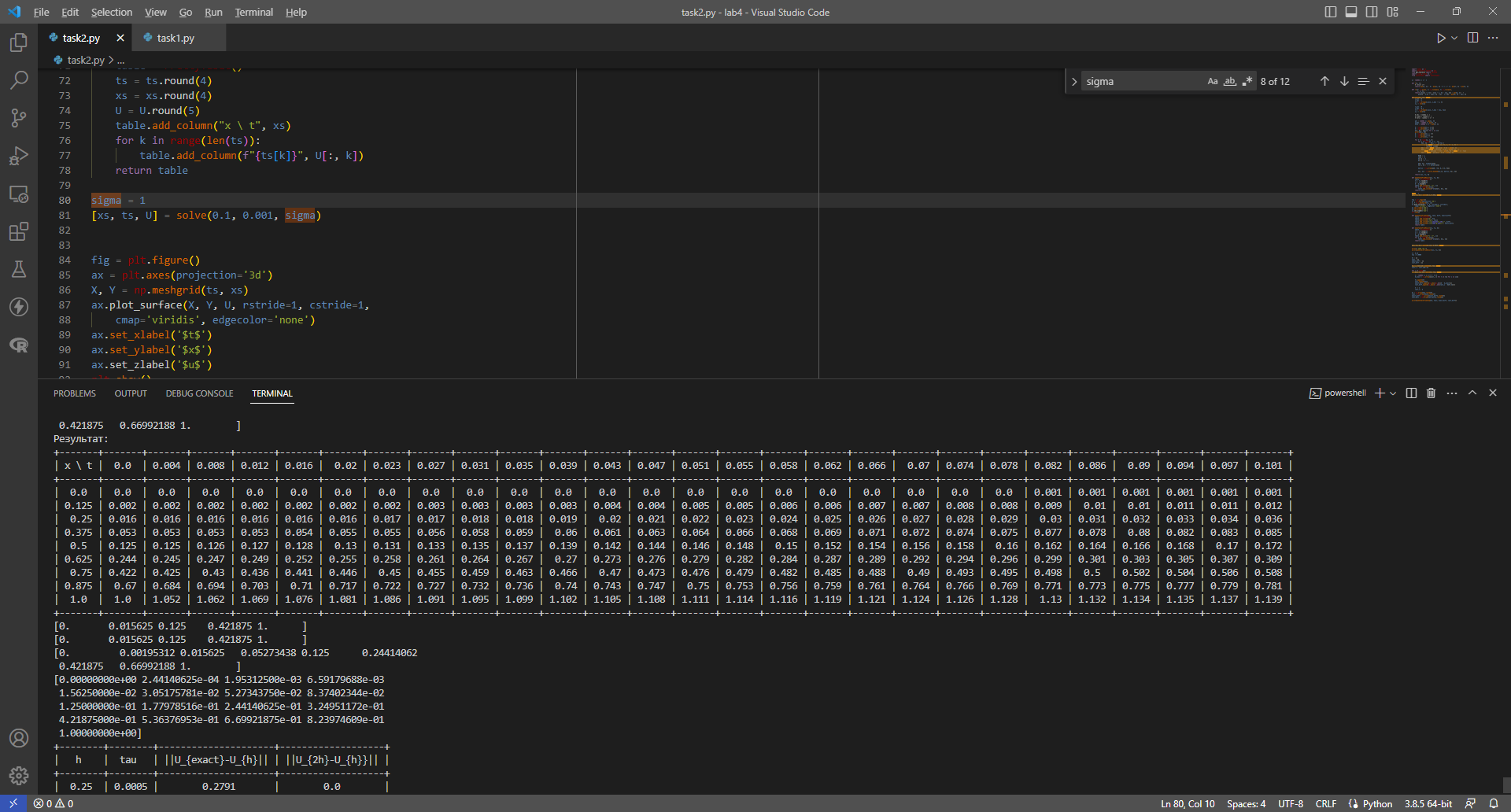
**Рисунок 12** Таблица, характеризующая точность решения для точной функции

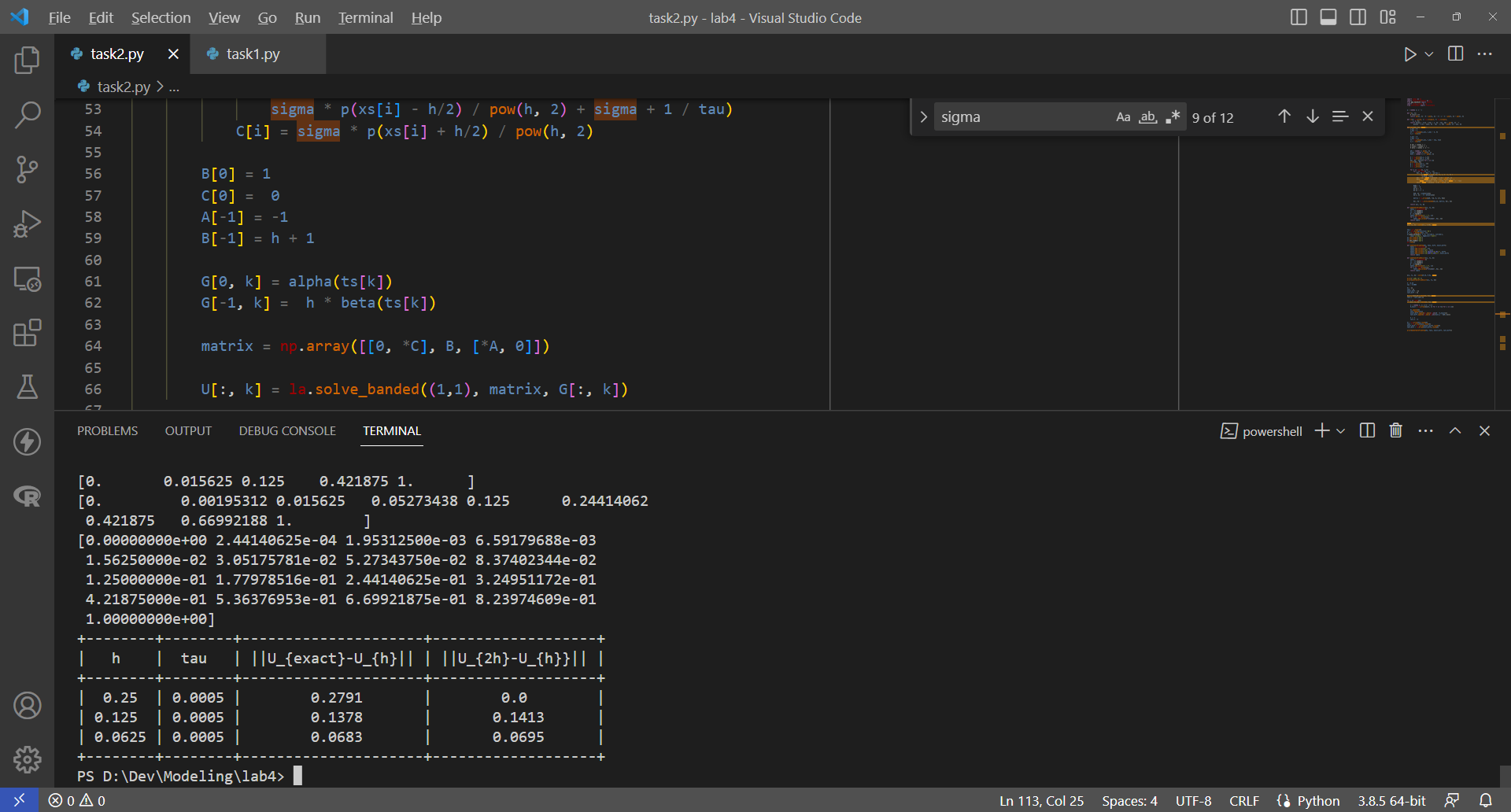
Аппроксимируем функцию :

Вычислим значения для функции при .



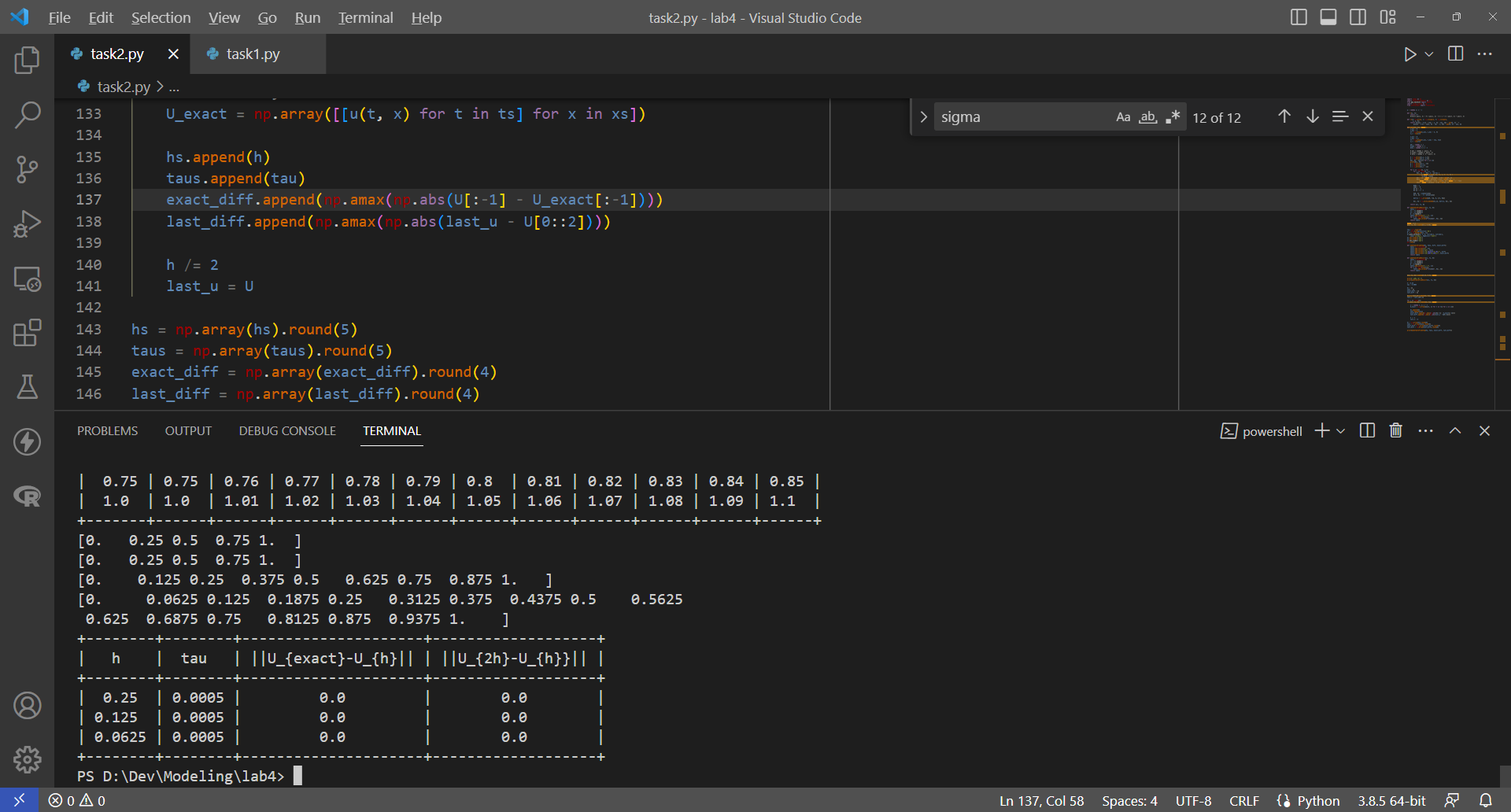
**Рисунок 13** Результат аппроксимации при



**Рисунок 14** Результат аппроксимации при 

**Рисунок 15** Таблица, характеризующая точность решения для аппроксимируемой функции

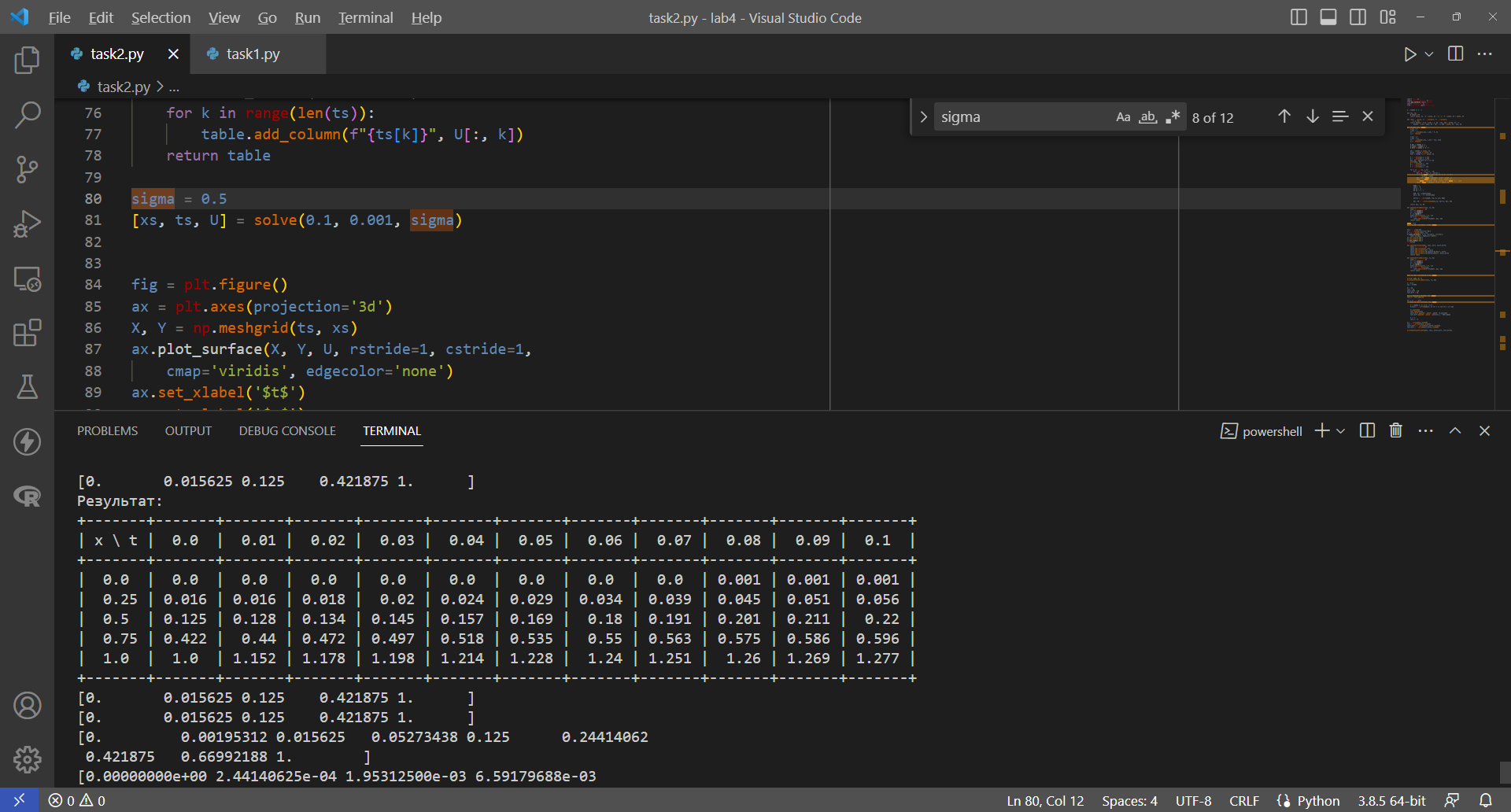
Проверим правильность для точной аппроксимации при   
.

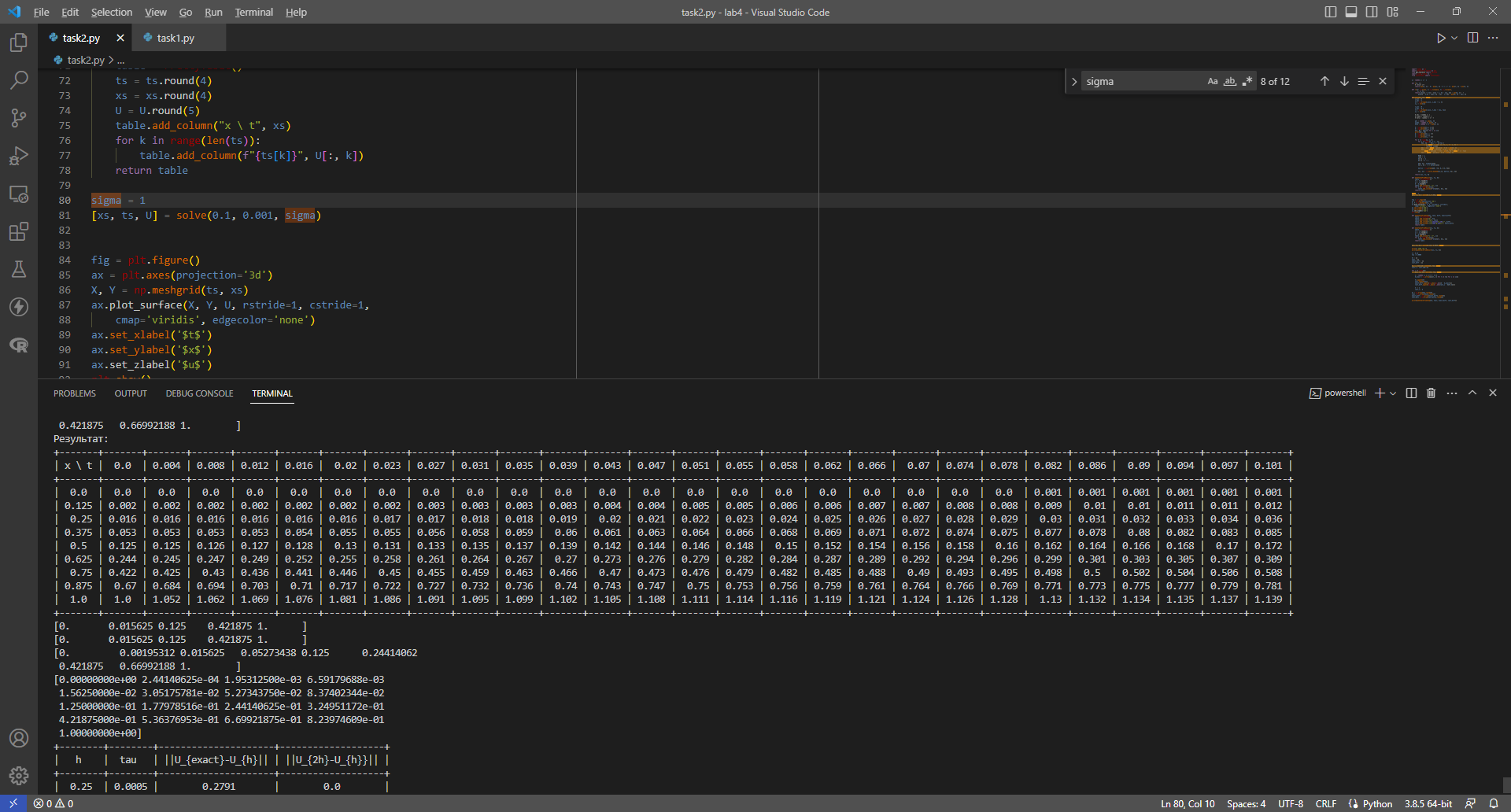


**Рисунок 16** Результат аппроксимации при

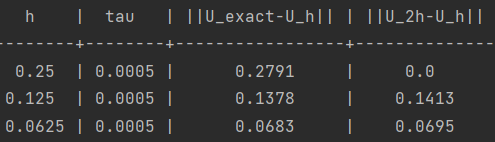
Аппроксимируем функцию :

Вычислим значения для функции при .

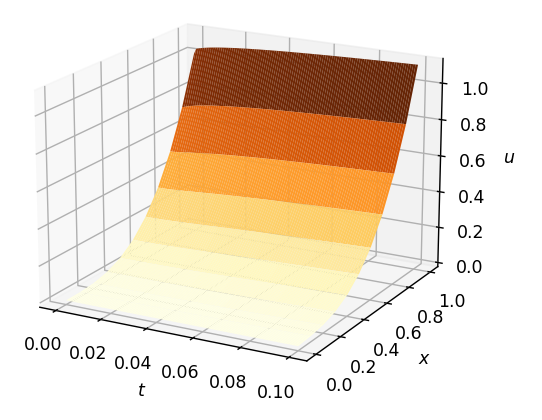


**Рисунок 17** Результат аппроксимации при 

**Рисунок 18** Результат аппроксимации при



**Рисунок 19** Таблица, характеризующая точность решения для аппроксимируемой функции



**Рисунок 20** График аппроксимированной функции

**Вывод**: в ходе выполнения работы были сформированы практические навыки анализа возможностей построения и выделения наиболее важных свойств объектов моделей для моделирования и использования специализированных программных пакетов и библиотек для стандартных вычислений и визуализации результатов численного или приближенно-аналитического решения ДУЧП2 параболического типа на основе сравнения результатов.

**ПРИЛОЖЕНИЯ**

**Листинг:**

**Задание №1**

import numpy as np

import matplotlib.pyplot as plt

from prettytable import PrettyTable

def f(x, t):

#return x\*\*2 - 4\*x - 1 + t

return pow(x, 3) - 9 \* pow(x, 2) - 6 \* x + 3 \* pow(t, 2) + pow(t, 3)

p = lambda x: x + 1

def lu(u: np.array, x: np.linspace, t: np.linspace,

i, k, h):

return p(x[i] + h/2) \* (u[i + 1, k] - u[i, k]) / pow(h, 2) - (

p(x[i] - h/2) \* (u[i, k] - u[i - 1, k]) / pow(h, 2)) - u[i, k]

def solve(h, tau):

x\_min = 0

x\_max = 1

xs = np.arange(x\_min, x\_max + h, h)

n\_x = len(xs)

t\_min = 0

t\_max = 0.1

ts = np.arange(t\_min, t\_max + tau, tau)

n\_t = len(ts)

#phi = lambda x, t: x\*\*2 + t

phi = lambda x, t: pow(x, 3)

#alpha = lambda t: t

alpha = lambda t: pow(t, 3)

#beta = lambda t: 3 + t

beta = lambda t: 1 + pow(t, 3) + 3

U = np.zeros((n\_x, n\_t))

U[:, 0] = [phi(x, t\_min) for x in xs]

U[0, 0] = alpha(ts[0])

U[-1, 0] = (2\*h\*beta(ts[0]) + 4\*U[-2, 0] - U[-3, 0])/(2\*h+3)

for k in range(0, n\_t - 1):

for i in range(1, n\_x - 1):

U[i, k + 1] = U[i, k] + tau \* (lu(U, xs, ts, i, k, h) +

f(xs[i], ts[k]))

U[0, k + 1] = alpha(ts[k+1])

U[-1, k + 1] = (2\*h\*beta(ts[k+1]) + 4\*U[-2, k+1] - U[-3, k+1])/(2\*h+3)

return [xs, ts, U]

def makeTableFromResult(xs, ts, U):

table = PrettyTable()

ts = ts.round(3)

xs = xs.round(3)

U = U.round(3)

table.add\_column("x \ t", xs)

for k in range(len(ts)):

table.add\_column(f"{ts[k]}", U[:, k])

return table

[xs, ts, U] = solve(0.0625, 0.01)

print("Результат:")

print(makeTableFromResult(xs, ts, U))

fig = plt.figure()

ax = plt.axes(projection='3d')

X, Y = np.meshgrid(ts, xs)

ax.plot\_surface(X, Y, U, rstride=1, cstride=1,

cmap='viridis', edgecolor='none')

ax.set\_xlabel('$t$')

ax.set\_ylabel('$x$')

ax.set\_zlabel('$u$')

plt.show()

def makeTableFromStep(hs, taus, exact\_diff, diff):

table = PrettyTable()

table.add\_column("h", hs)

table.add\_column("tau", taus)

table.add\_column("||U\_{exact}-U\_{h}||", diff)

table.add\_column("||U\_{2h}-U\_{h}}||", exact\_diff)

return table

h = 0.25

tau = 0.001

hs = []

taus = []

exact\_diff = []

last\_diff = []

[\_, \_, last\_u] = solve(h, tau)

last\_u = last\_u[0::2]

for i in range(3):

[xs, ts, U] = solve(h, tau)

u = lambda t, x: x\*\*3 + t\*\*3

U\_exact = np.array([[u(t, x) for t in ts] for x in xs])

hs.append(h)

taus.append(tau)

exact\_diff.append(np.amax(np.abs(U - U\_exact)))

last\_diff.append(np.amax(np.abs(last\_u - U[0::2])))

h /= 2

last\_u = U

hs = np.array(hs).round(5)

taus = np.array(taus).round(5)

exact\_diff = np.array(exact\_diff).round(5)

last\_diff = np.array(last\_diff).round(5)

print(makeTableFromStep(hs, taus, last\_diff, exact\_diff))

**Задание №2**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits import mplot3d

import scipy.linalg as la

from prettytable import PrettyTable

p = lambda x: x + 1

def f(x, t):

# return x+t

return pow(x, 3) - 9 \* pow(x, 2) - 6 \* x + 3 \* pow(t, 2) + pow(t, 3)

def lu(u: np.array, x: np.linspace, t: np.linspace,

i, k, h):

return p(x[i] + h/2) \* (u[i + 1, k] - u[i, k]) / pow(h, 2) - \

p(x[i] - h/2) \* (u[i, k] - u[i - 1, k]) / pow(h, 2) - u[i, k]

def solve(h, tau, sigma):

x\_min = 0

x\_max = 1

xs = np.arange(x\_min, x\_max + h, h)

n\_x = len(xs)

t\_min = 0

t\_max = 0.1

ts = np.arange(t\_min, t\_max + tau, tau)

n\_t = len(ts)

# phi = lambda x: x

# alpha = lambda t: t

# beta = lambda t: 2 + t

phi = lambda x: pow(x, 3)

alpha = lambda t: pow(t, 3)

beta = lambda t: 4 + pow(t, 3)

U = np.zeros((n\_x, n\_t))

G = np.zeros((n\_x, n\_t))

U[:, 0] = [phi(x) for x in xs]

print(U[:, 0])

A = np.zeros((n\_x - 1))

B = np.zeros((n\_x))

C = np.zeros((n\_x - 1))

for k in range(1, n\_t):

for i in range(1, n\_x - 1):

G[i, k] = - U[i, k - 1]/tau \

- (1-sigma) \* lu(U, xs, ts, i, k - 1, h) \

- f(xs[i], ts[k])

A[i - 1] = sigma \* p(xs[i] - h/2) / pow(h, 2)

B[i] = - (sigma \* p(xs[i] + h/2) / pow(h, 2) + \

sigma \* p(xs[i] - h/2) / pow(h, 2) + sigma + 1 / tau)

C[i] = sigma \* p(xs[i] + h/2) / pow(h, 2)

B[0] = 1

C[0] = 0

A[-1] = -1

B[-1] = h + 1

G[0, k] = alpha(ts[k])

G[-1, k] = h \* beta(ts[k])

matrix = np.array([[0, \*C], B, [\*A, 0]])

U[:, k] = la.solve\_banded((1,1), matrix, G[:, k])

return [xs, ts, U]

def makeTableFromResult(xs, ts, U):

table = PrettyTable()

ts = ts.round(4)

xs = xs.round(4)

U = U.round(5)

table.add\_column("x \ t", xs)

for k in range(len(ts)):

table.add\_column(f"{ts[k]}", U[:, k])

return table

sigma = 0.5

[xs, ts, U] = solve(0.1, 0.001, sigma)

fig = plt.figure()

ax = plt.axes(projection='3d')

X, Y = np.meshgrid(ts, xs)

ax.plot\_surface(X, Y, U, rstride=1, cstride=1,

cmap='viridis', edgecolor='none')

ax.set\_xlabel('$t$')

ax.set\_ylabel('$x$')

ax.set\_zlabel('$u$')

plt.show()

def makeTableFromStep(hs, taus, diff, exact\_diff):

table = PrettyTable()

table.add\_column("h", hs)

table.add\_column("tau", taus)

table.add\_column("||U\_{exact}-U\_{h}||", diff)

table.add\_column("||U\_{2h}-U\_{h}}||", exact\_diff)

return table

def makeTableFromResult(xs, ts, U):

table = PrettyTable()

ts = ts.round(3)

xs = xs.round(3)

U = U.round(3)

table.add\_column("x \ t", xs)

for k in range(len(ts)):

table.add\_column(f"{ts[k]}", U[:, k])

return table

[xs, ts, U] = solve(0.25, 0.01, sigma)

print("Результат:")

print(makeTableFromResult(xs, ts, U))

h = 0.25

tau = 0.0005

hs = []

taus = []

exact\_diff = []

last\_diff = []

[\_, \_, last\_u] = solve(h, tau, sigma)

last\_u = last\_u[0::2]

for i in range(3):

[xs, ts, U] = solve(h, tau, sigma)

u = lambda t, x: x\*\*3 + t\*\*3

U\_exact = np.array([[u(t, x) for t in ts] for x in xs])

hs.append(h)

taus.append(tau)

exact\_diff.append(np.amax(np.abs(U - U\_exact)))

last\_diff.append(np.amax(np.abs(last\_u - U[0::2])))

h /= 2

last\_u = U

hs = np.array(hs).round(5)

taus = np.array(taus).round(5)

exact\_diff = np.array(exact\_diff).round(4)

last\_diff = np.array(last\_diff).round(4)

print(makeTableFromStep(hs, taus, exact\_diff, last\_diff))