Второе задание по вычислительной математике. Гиперболические системы уравнений

Рассматривается задача

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{A}\frac{\partial \mathbf{u}}{\partial x} = \mathbf{b}(x), 0 \leqslant x \leqslant 1, 0 \leqslant t \leqslant 1, \quad \mathbf{u}(x,0) = \begin{pmatrix} x^3 \\ 1 - x^2 \\ x^2 + 1 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} -4 & \frac{3}{5} & \frac{128}{5} \\ -1 & \frac{-36}{5} & \frac{-16}{5} \\ 1 & \frac{1}{5} & \frac{-19}{5} \end{pmatrix}, \mathbf{b}(x) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Часть 1. Приведение системы к характеристическому виду

Выполним спектральное разложение матрицы A системы:

$$A = PDP^{-1}$$

$$A = \begin{pmatrix} -4 & \frac{3}{5} & \frac{128}{5} \\ -1 & \frac{-36}{5} & \frac{-16}{5} \\ 1 & \frac{1}{5} & \frac{-19}{5} \end{pmatrix} = \begin{pmatrix} -5 & \frac{-1}{5} & 5 \\ -1 & 1 & -1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} -9 & 0 & 0 \\ 0 & -7 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -\frac{1}{10} & \frac{-1}{50} & \frac{12}{25} \\ 0 & 1 & 1 \\ \frac{1}{10} & \frac{1}{50} & \frac{13}{25} \end{pmatrix}$$

С учётом разложения система принимает вид:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{PDP^{-1}}\frac{\partial \mathbf{u}}{\partial x} = 0$$

Домножая слева на P^{-1}, получим систему в характеристическом виде:

$$\frac{\partial \mathbf{R}}{\partial t} + \mathbf{D}\frac{\partial \mathbf{R}}{\partial x} = 0$$

где $\mathbf{R} = \mathbf{P^{-1}u}$

В итоге получаем следующую систему уравнений:

$$\begin{cases} \frac{\partial R_1}{\partial t} - 9\frac{\partial R_1}{\partial x} = 0 \\ \frac{\partial R_2}{\partial t} - 7\frac{\partial R_2}{\partial x} = 0 \\ \frac{\partial R_3}{\partial t} + 1\frac{\partial R_3}{\partial x} = 0 \end{cases}$$

Начальные условия для инвариантов Римана:

$$\mathbf{R}(x,0) = \begin{pmatrix} \frac{-5x^3+25x^2+23}{50} \\ 2 \\ \frac{5x^3+25x^2+27}{50} \end{pmatrix}$$

Восстановление естественных переменных возможно по формуле:

$$\mathbf{u} = \mathbf{PR}$$

Часть 2. Схема Куранта-Изаксона-Риса

Для первого и второго уравнения системы применима схема:

$$\frac{u_m^{n+1} - u_m^n}{\tau} + a\frac{u_m^n - u_{m-1}^n}{h} = 0$$

Перепишем её для третьего уравнения, где $a < 0$:

$$\frac{u_m^{n+1} - u_m^n}{\tau} + a\frac{u_{m+1}^n - u_m^n}{h} = 0$$

Согласно определению Фридрихса, данные схемы не являются монотонными. Первое дифференциальное приближение запишется следующим образом:

$$r_{\tau h} = Lu - \frac{ch}{2}\left(1 - \frac{c\tau}{h}\right)\frac{\partial^2 u}{\partial x^2} + O\left(\tau^2 + h^2\right)$$

Можно видеть, что преобладает диссипативная ошибка. А теперь перейдём к реализации схемы:

In [310]:

```
import abc
import numpy as np

class TwoLayerDifferenceScheme(object):
    def __init__(self, t0, te, x0, xe, xsteps, tsteps, init: callable):
        self.t = t0
        self.te = te
        self.xgrid = np.linspace(x0, xe, xsteps + 1)
        self.h = self.xgrid[1] - self.xgrid[0]
        self.tgrid = np.linspace(t0, te, tsteps + 1)
        self.tau = self.tgrid[1] - self.tgrid[0]
        self.u_current = init(self.t, self.xgrid)

    @abc.abstractmethod
    def step(self):
        raise NotImplementedError

    def solve(self):
        yield self.u_current
        while self.t < self.te:
            self.step()
            yield self.u_current
```

In [311]:

```
class KIRLeft(TwoLayerDifferenceScheme):
    def __init__(self, t0, te, x0, xe, xsteps, tsteps, init: callable, a):
        super(KIRLeft, self).__init__(t0, te, x0, xe, xsteps, tsteps, init)
        assert a > 0
        self.a = a
    def step(self):
        u = np.zeros_like(self.u_current)
        u[0] = self.u_current[0]
        for i in range(1, len(u)):
            u[i] = self.u_current[i]-(self.tau*self.a/self.h)*(self.u_current[i]-self.u_cur
        self.u_current = u
        self.t += self.tau

class KIRRight(TwoLayerDifferenceScheme):
    def __init__(self, t0, te, x0, xe, xsteps, tsteps, init: callable, a):
        super(KIRRight, self).__init__(t0, te, x0, xe, xsteps, tsteps, init)
        assert a < 0
        self.a = a
    def step(self):
        u = np.zeros_like(self.u_current)
        u[-1] = self.u_current[-1]
        for i in range(0, len(u)-1):
            u[i] = self.u_current[i]-(self.tau*self.a/self.h)*(self.u_current[i+1]-self.u_c
        self.u_current = u
        self.t += self.tau
```

Реализованы две схемы КИР: с правым уголком для $a < 0$, и с левым для $a > 0$. Получим решения системы в инвариантах Римана:

In [363]:

```
first = KIRRight(0,1,0,1,50,1000,lambda t, x: 0.02*(-5*x**3+25*x**2+23), -9)
R_1 = np.array(list(first.solve()))
second = KIRRight(0,1,0,1,50,1000,lambda t, x: (0*x**3+0*x**2+2), -7)
R_2 = np.array(list(second.solve()))
third = KIRLeft(0,1,0,1,50,1000,lambda t, x: 0.02*(5*x**3+25*x**2+27), 1)
R_3 = np.array(list(third.solve()))
```

Вернёмся к естественным переменным при помощи преобразования, задаваемого матрицей $\mathbf{P}$:

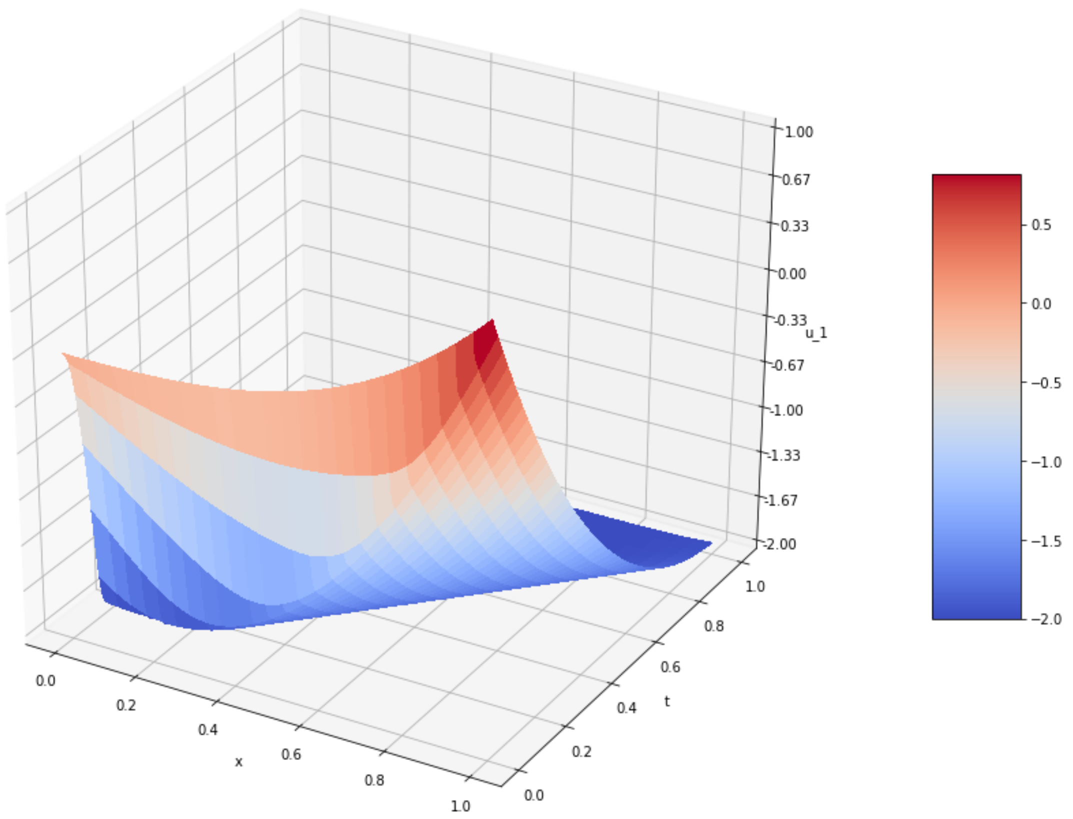
In [364]:

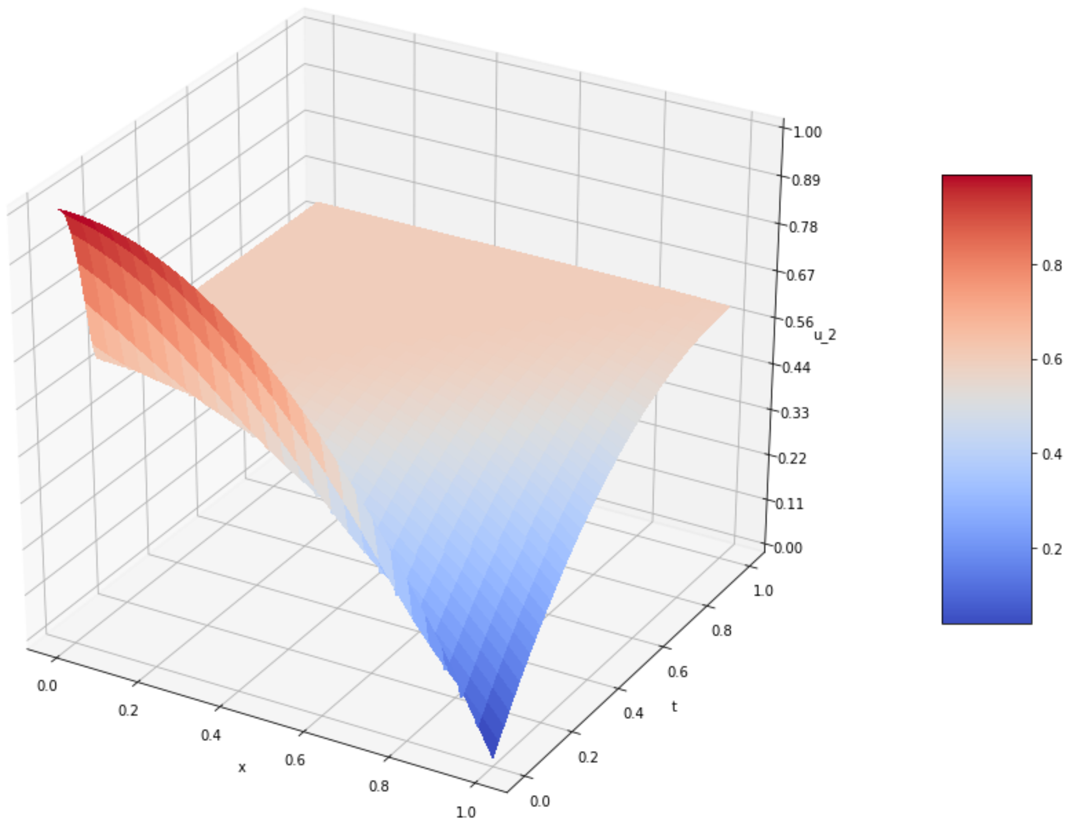
```
u_1 = (-5)*np.array(R_1) + (-1/5)*np.array(R_2) + 5*np.array(R_3)
u_2 = (-1)*np.array(R_1) + 1*np.array(R_2) + (-1)*np.array(R_3)
u_3 = 1*np.array(R_1) + 0*np.array(R_2) + 1*np.array(R_3)
```

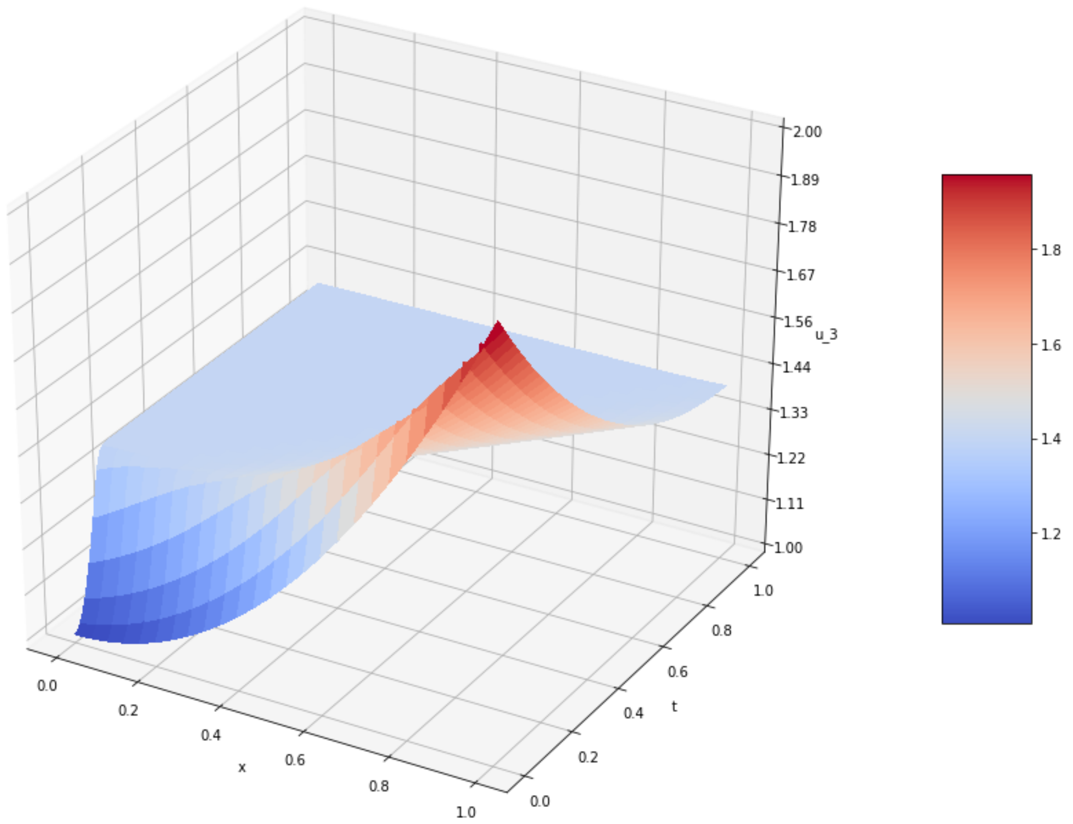
Визуализируем полученные результаты:

In [365]:

```python
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt

def plot_3d(u, label):
    fig = plt.figure(figsize=(16,12))
    ax = fig.gca(projection='3d')
    ts = first.tgrid
    xs = first.xgrid
    X, Y = np.meshgrid(xs, ts)
    surf = ax.plot_surface(X, Y, u, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
    ax.set_xlabel("x")
    ax.set_ylabel("t")
    ax.set_zlabel(label)
    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.show()

plot_3d(u_1, 'u_1')
plot_3d(u_2, 'u_2')
plot_3d(u_3, 'u_3')
```







Часть 2. Схема Федоренко

```
\begin{equation}
```

```
\frac{u_{m}^{n+1}-u_{m}^{n}}{\tau}+\frac{u_{m}^{n}-u_{m-1}^{n}}{h}+\frac{\gamma}{2
\tau}\left(\frac{a \tau}{h}-\frac{a^{2} \tau^{2}}{h^{2}}\right)\left(u_{m+1}^{n}-2
u_{m}^{n}+u_{m-1}^{n}\right)=0
\end{equation}

\begin{equation}
\gamma=\left\{\begin{array}{ll}{1,} & {\left|y_{m-1}^{n}-2 y_{m}^{n}+y_{m+1}^{n}\right|
<\xi\left|y_{m}^{n}-y_{m-1}^{n}\right|} \\ {0,} & {\left|y_{m-1}^{n}-2
y_{m}^{n}+y_{m+1}^{n}\right|>\xi\left|y_{m}^{n}-y_{m-1}^{n}\right|}\end{array}\right.
\end{equation}
```

Данная схема при обращении анализатора гладкости решения в ноль переходит в схему Куранта-Изаксона-Риса. Всё, что сказано о ней в предыдущей части остаётся справедливым. В случае $\gamma = 1$ получаем схему второго порядка аппроксимации. В первом дифференциальном приближении (в данном отчёте не приводится ввиду громоздкости выкладок) находятся члены с третьей производной, что говорит о преобладании дисперсионной ошибки. Согласно определению Фридрихса, схема не является монотонной.

В отлчие от предыдущего случая, для использования данного метода придётся задать граничные условия для обоих концов рассчётного отрезка. Поэтому решения, полученные при помощи предыдущих схем и схемы Федоренко, будут качественно отличаться друг от друга. Сделав это замечание, перейдём к реализации схемы:

In [233]:

```python
class Fedorenko(TwoLayerDifferenceScheme):
    def __init__(self, t0, te, x0, xe, xsteps, tsteps, init: callable, a, xhi):
        super(Fedorenko, self).__init__(t0, te, x0, xe, xsteps, tsteps, init)
        assert xhi >= 0
        self.xhi = xhi
        self.a = a
    def step(self):
        u = np.zeros_like(self.u_current)
        u[0] = self.u_current[0]
        u[-1] = self.u_current[-1]
        for i in range(1, len(u) - 1):
            gamma = abs(self.u_current[i-1]-2*self.u_current[i]+self.u_current[i+1]) < self
            u[i] = self.u_current[i]-(self.tau/self.h)*(self.u_current[i]-self.u_current[i-
        self.u_current = u
        self.t += self.tau
```

In [258]:

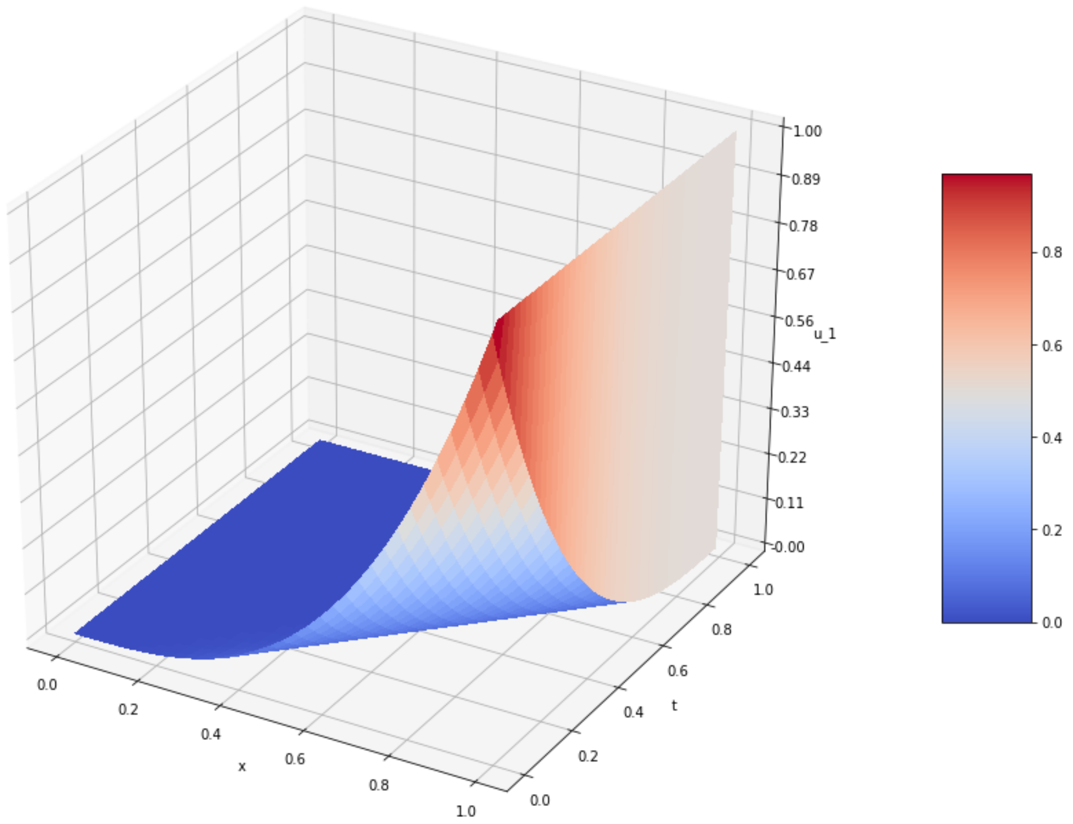
```python
xhi = 0

first = Fedorenko(0,1,0,1,100,500,lambda t, x: 0.02*(-5*x**3+25*x**2+23), -9, xhi)
R_1 = np.array(list(first.solve()))
second = Fedorenko(0,1,0,1,100,500,lambda t, x: (0*x**3+0*x**2+2), -7, xhi)
R_2 = np.array(list(second.solve()))
third = Fedorenko(0,1,0,1,100,500,lambda t, x: 0.02*(5*x**3+25*x**2+27), 1, xhi)
R_3 = np.array(list(third.solve()))
```

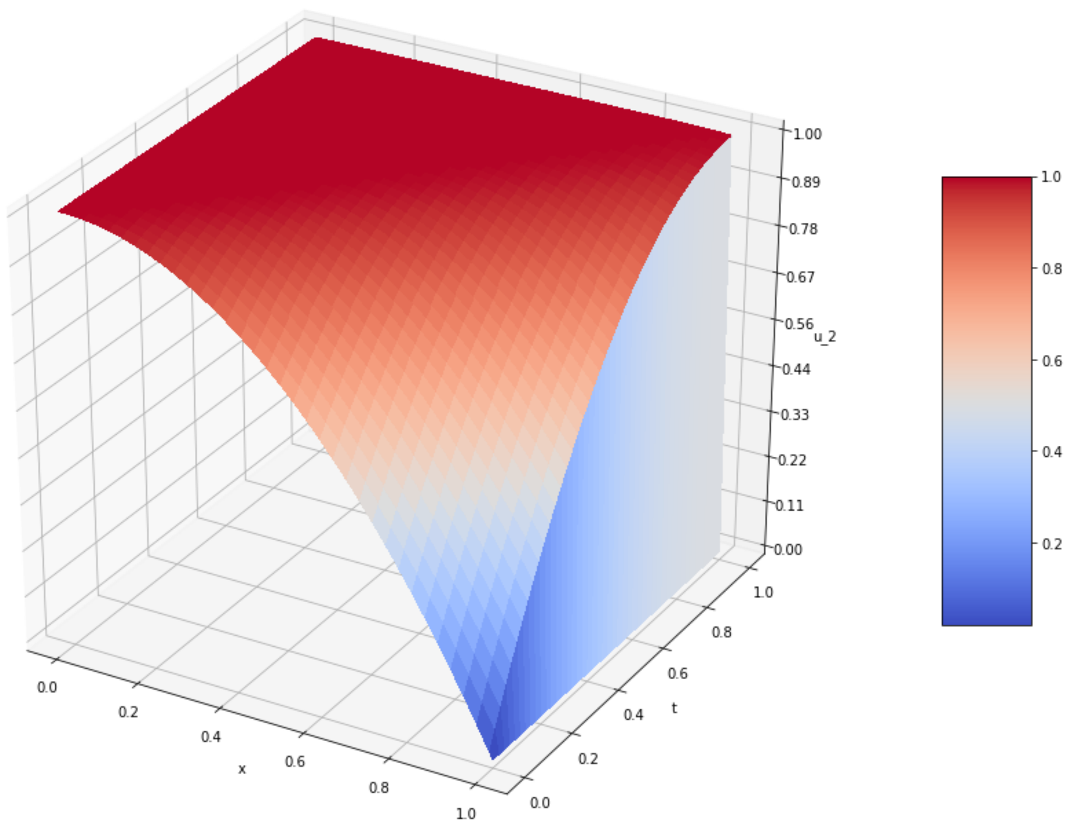
In [259]:

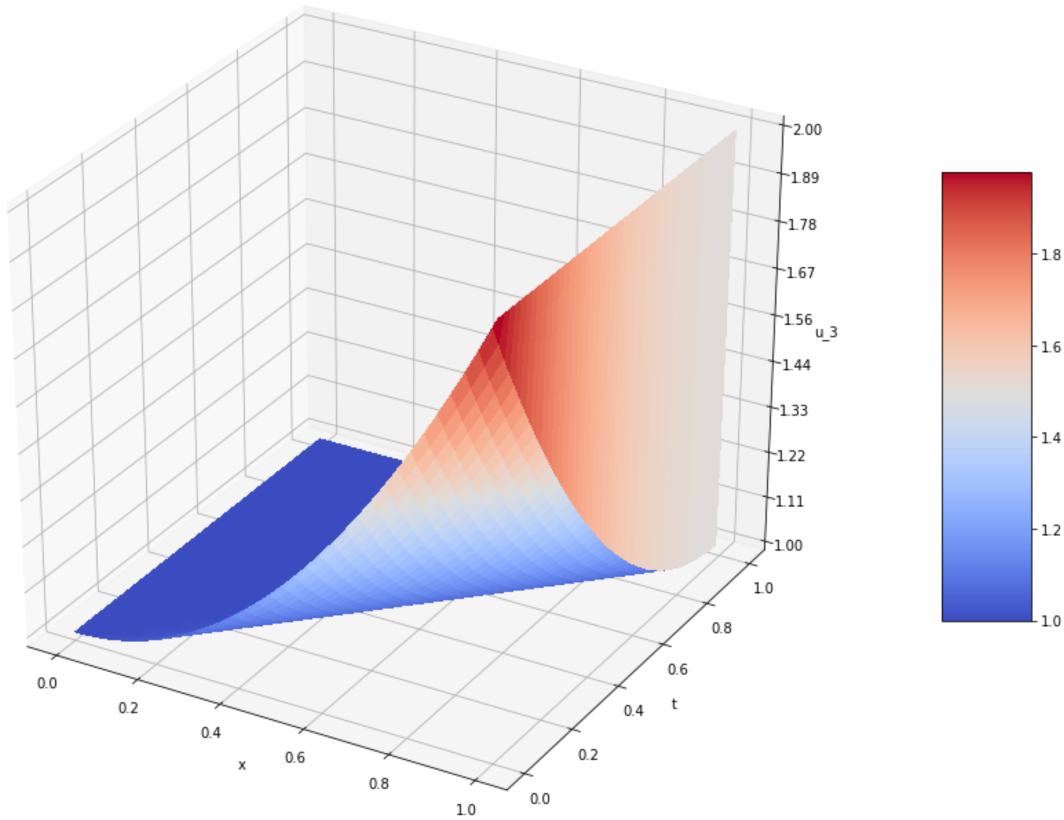
```python
u_1 = (-5)*np.array(R_1) + (-1/5)*np.array(R_2) + 5*np.array(R_3)
u_2 = (-1)*np.array(R_1) + 1*np.array(R_2) + (-1)*np.array(R_3)
u_3 = 1*np.array(R_1) + 0*np.array(R_2) + 1*np.array(R_3)
```

In [260]:

```
plot_3d(u_1, 'u_1')
plot_3d(u_2, 'u_2')
plot_3d(u_3, 'u_3')
```







В рассматриваемом примере при изменении параметра ξ не происходит существенных изменений внешнего вида решений