Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №13

По дисциплине «Методы численного анализа»

По теме «Метод сеток решения одномерного нестационарного уравнения теплопроводности»

Вариант 6

Выполнил:

студент гр. 653504

Куликов А.Д.

Проверил:

Анисимов В.Я.

Минск 2018

**Краткие теоретические сведения**

**Постановка задачи:**

Требуется найти непрерывную на замкнутом прямоугольнике D функцию u(x, t), которая на D′ удовлетворяет уравнению теплопроводности:



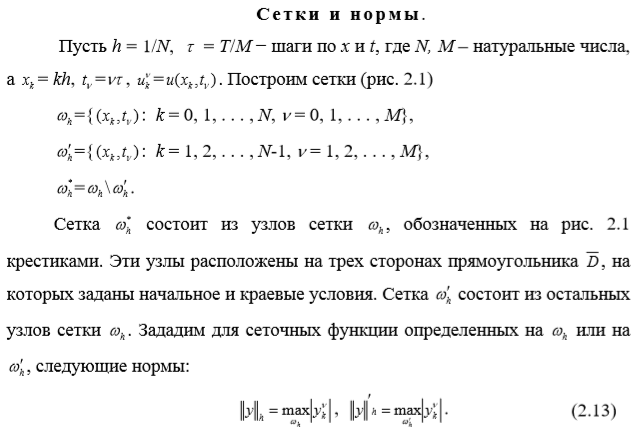
которое при t = 0 удовлетворяет начальному условию

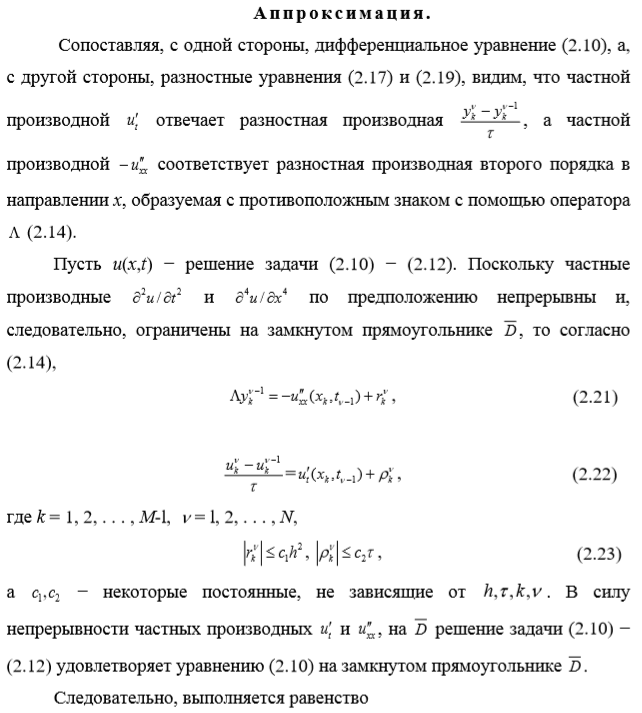


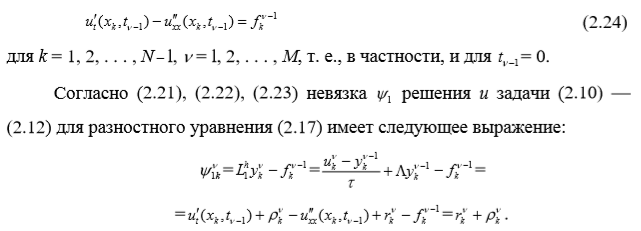
а при х = 0 и х = 1 подчиняется краевым условиям

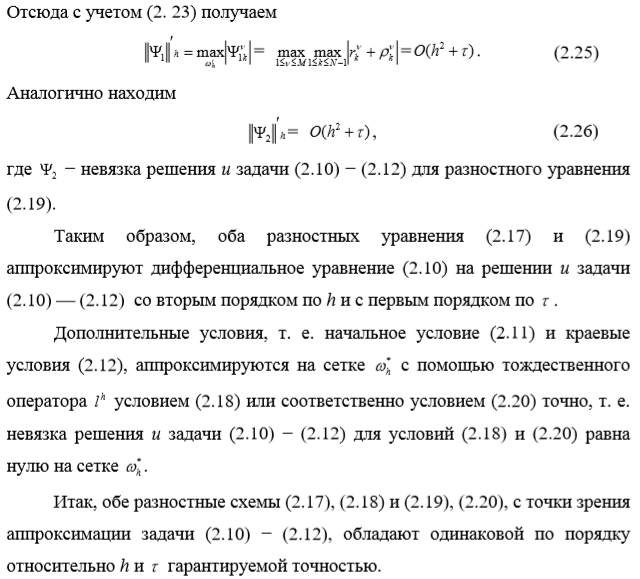


где f(x, t), s(x), p(t), q(t) − заданные достаточно гладкие функции, причем s(0) = p(0), s(l) = q(l). Задача (2.10) − (2. 12) называется смешанной задачей, поскольку она содержит как начальные условия, так и краевые условия. Известно, (11) что у поставленной задачи существует единственное решение u(х, t). Мы будем предполагать, что это решение имеет на замкнутом прямоугольнике D непрерывные частные производные 

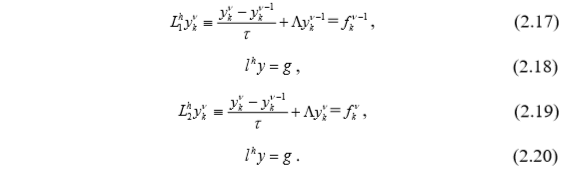


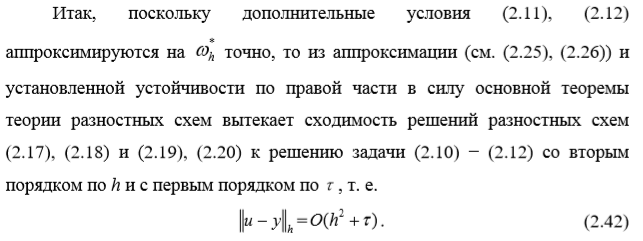


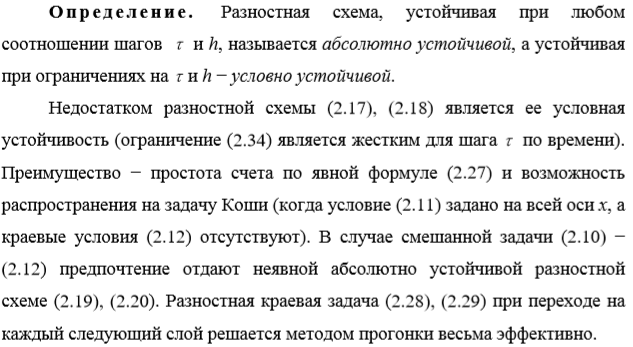




**Устойчивость и сходимость**

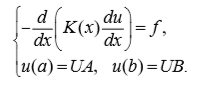






**Задание 1**

**Промоделировать стационарные процессы теплопроводности стержня в зависимости от входных данных задачи:**



Исходные данные:

**Решение**

Для нахождения решения стационарного уравнения теплопроводности

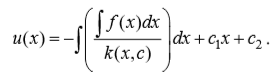


дважды проинтегрируем его.

В результате первого интегрирования получим:



После второго интегрирования:



Значения найдем из граничных условий:

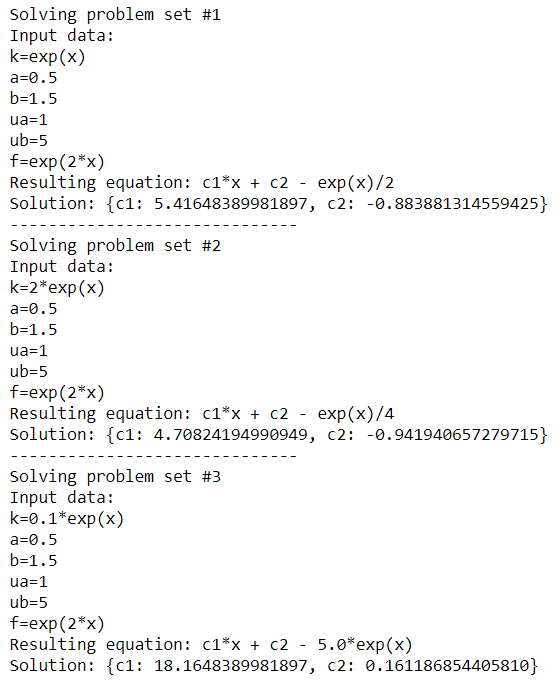
Подставив исходные данные, получим:

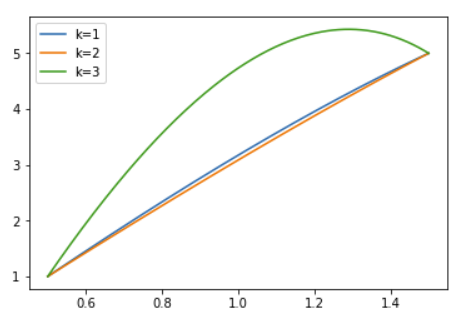
Таблица набора параметров:



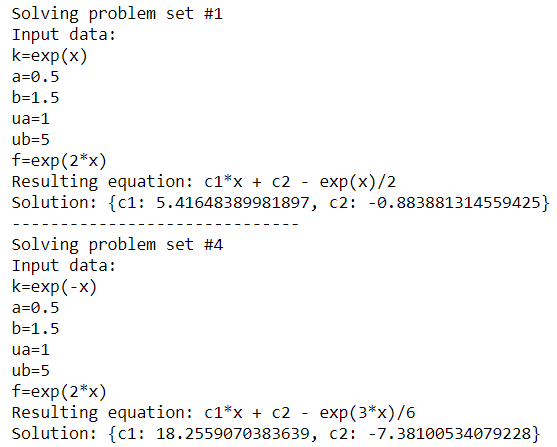
**Результат работы программы**

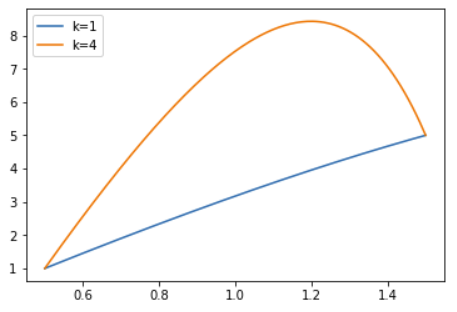
А) Построим решения для наборов параметров 1-3:



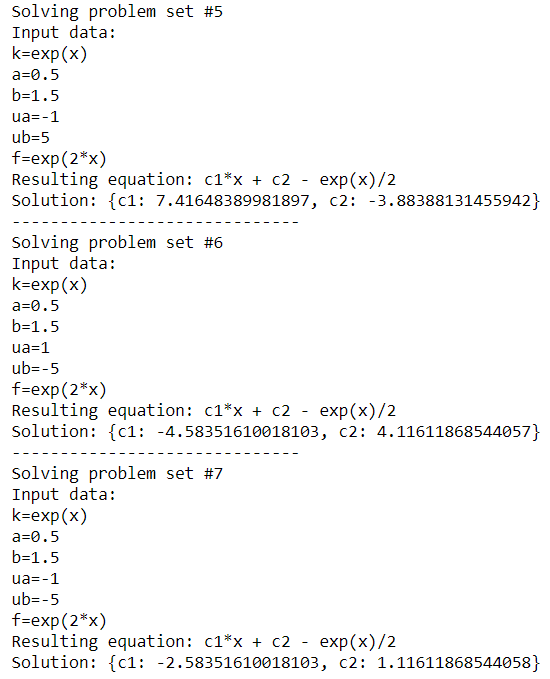


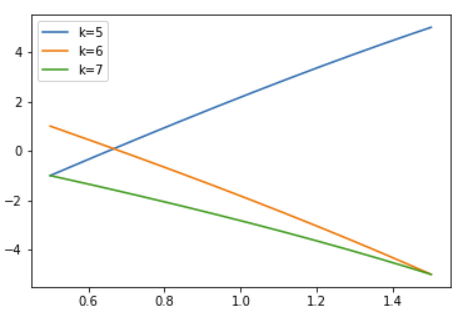
Б) Построим решения для наборов параметров 1 и 4:





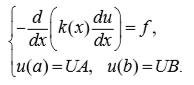
В) Построим решения для наборов параметров 5-7

:



**Задание 2**

**Промоделировать стационарные процессы теплопроводности стержня в зависимости от входных данных задачи – переменного коэффициента теплопроводности k(x) и плотности источников тепла f(x):**



Исходные данные:

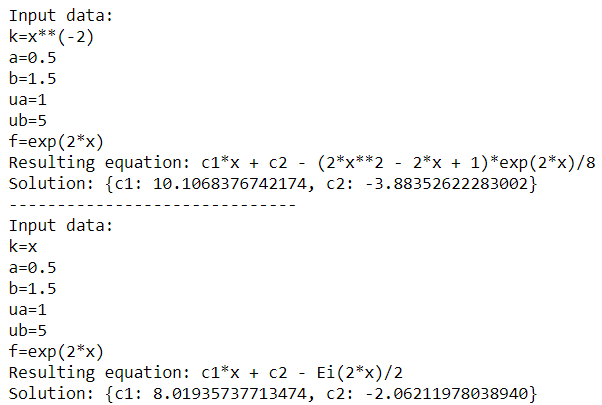
**Решение**

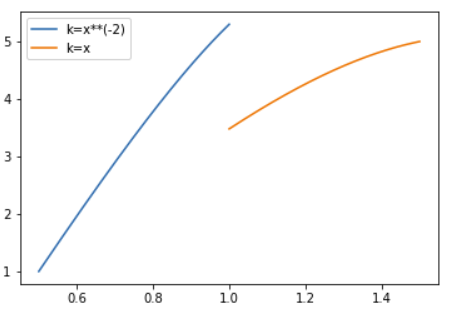
**А) Полагать, что стержень состоит из 2-x материалов с различными k(x):**



Возьмем ;

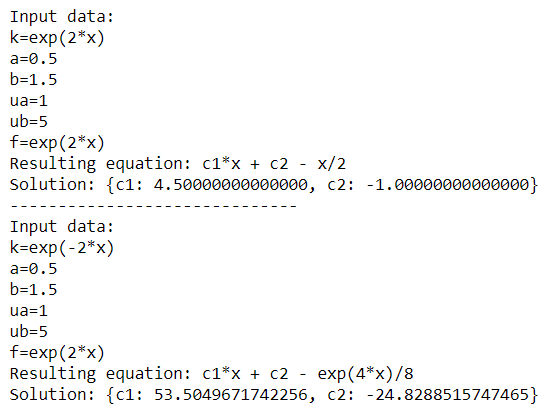
Результат работы программы:

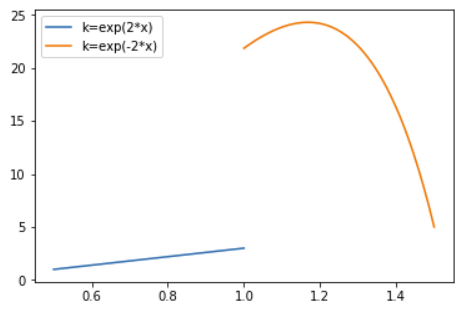




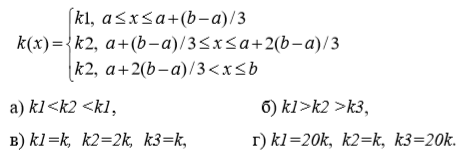
Пусть

Результат работы программы:

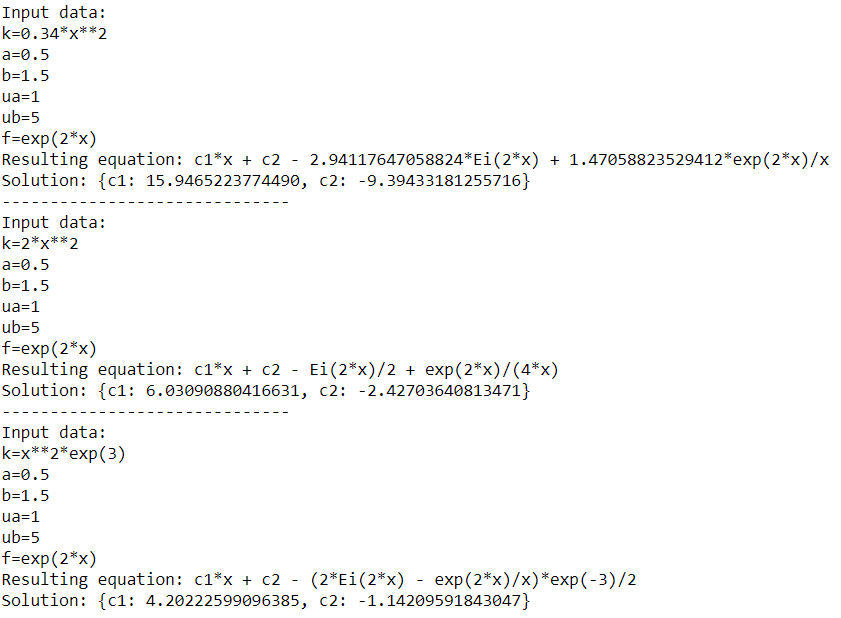


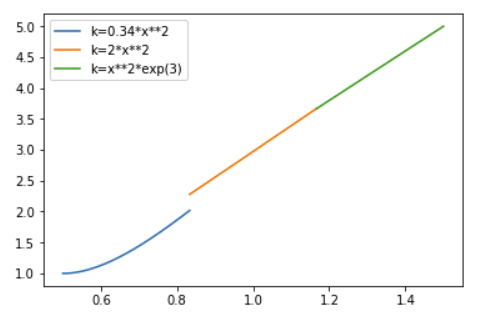


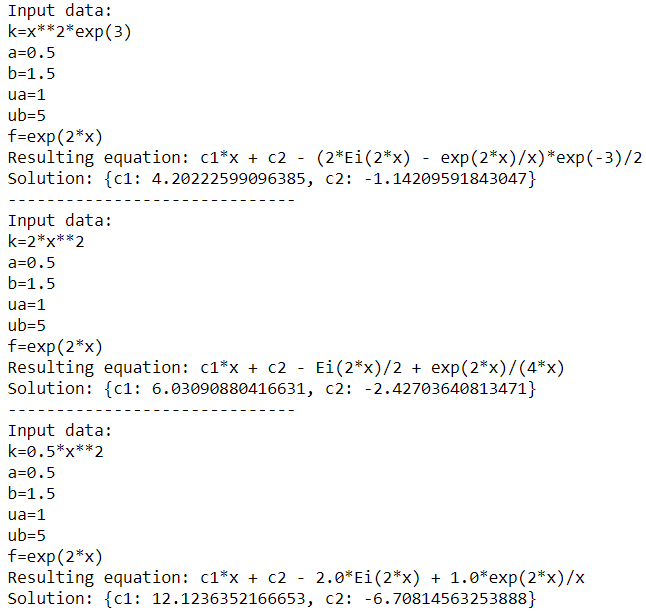
**Б) Пусть стержень состоит из 3-x материалов с различными свойствами:**

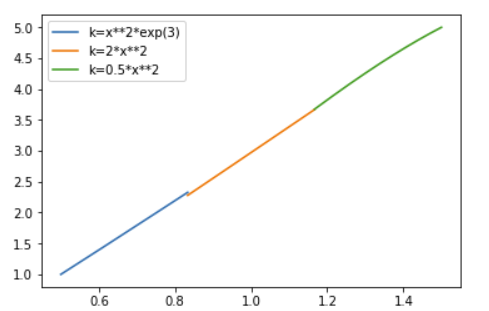


**Результат работы программы**

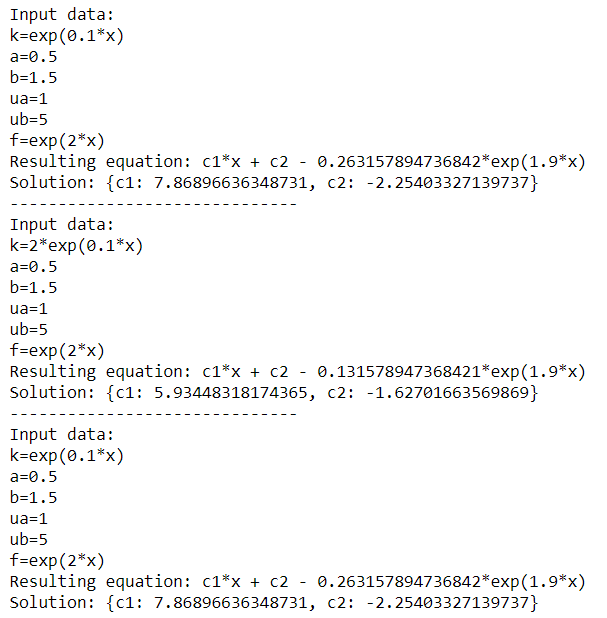


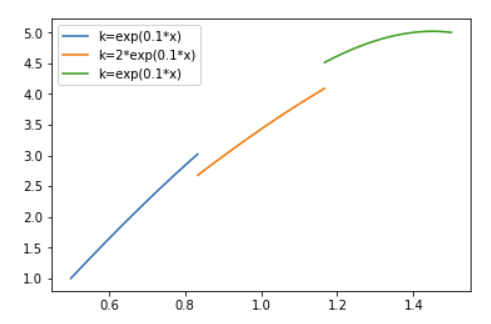




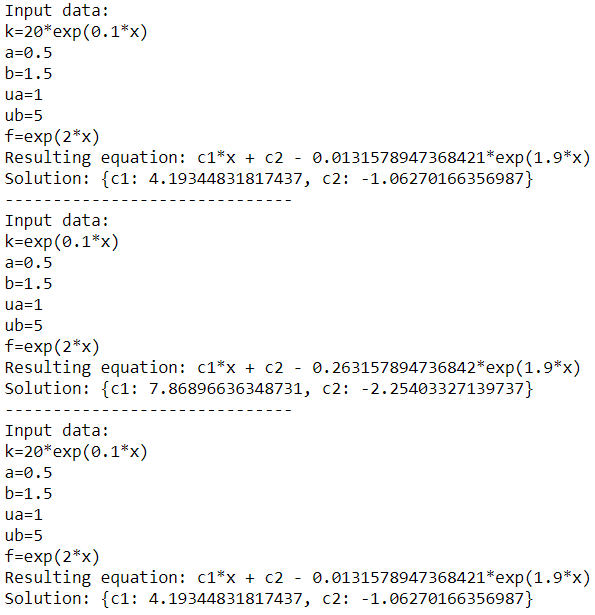


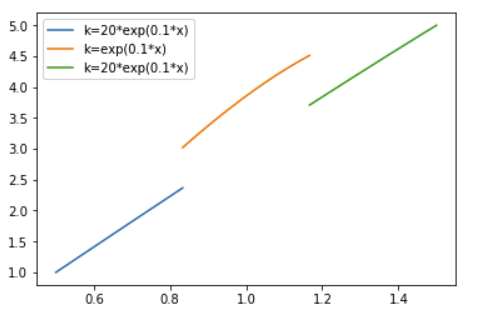
1. :





1. :





**В) Промоделировать процесс теплопроводности в зависимости от правой части – функции , предполагая, что - точечный источник тепла. Задать точечный источник тепла можно следующим образом: где c- некоторая константа (мощность источника), - дельта-функция, - точка из отрезка в которой располагается источник.**

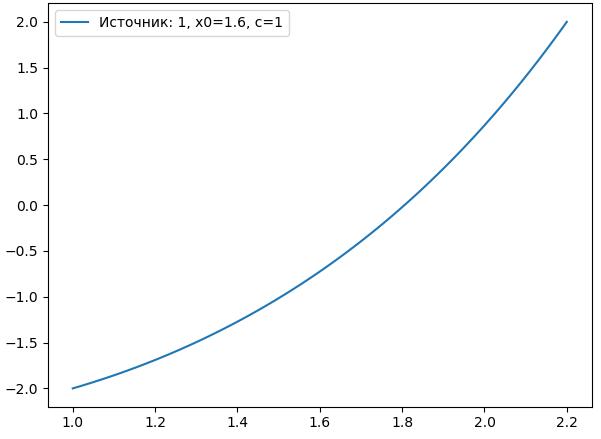
**а) точечный источник поставлен в середину отрезка [a, b]:**

x0 = (2.2 + 1) / 2 = 1.6 – середина отрезка [a, b]

f(x) = c\*δ(x – x0) = δ(x – 1.6)

**Результат работы программы**



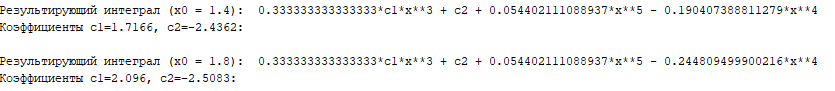


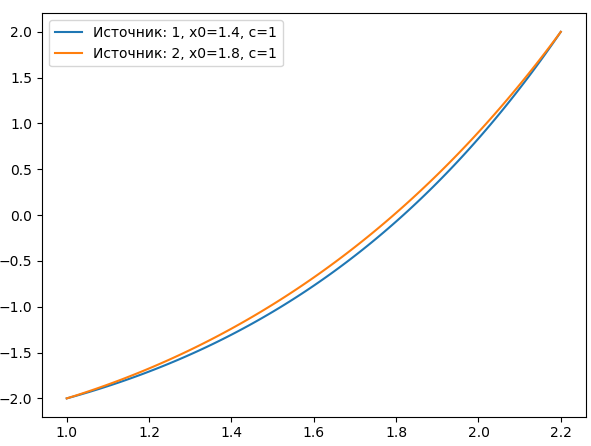
**б) два одинаковых по мощности источника поставлены в разные точки отрезка, симметричные относительно середины отрезка:**

(x) = δ(x – 1.4)

(x) = δ(x – 1.8)

**Результат работы программы**



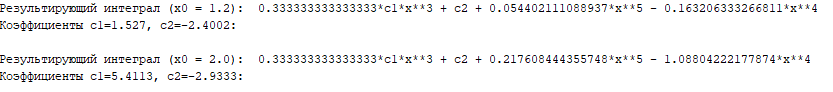


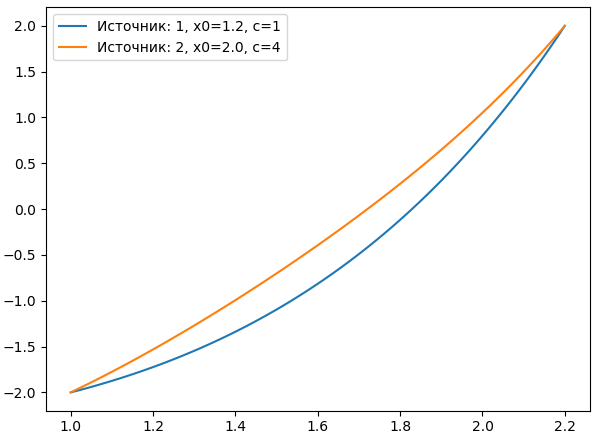
**в) два различных по мощности источника поставлены симметрично:**

(x) = δ(x – 1.2)

(x) = 4\*δ(x – 2)

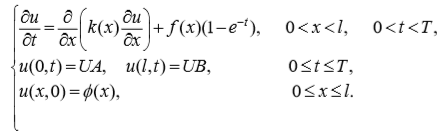
**Результат работы программы**





**Задание 3**

**Промоделировать нестационарные процессы теплопроводности в зависимости от входных данных задачи - коэффициента теплопроводности k(x) и начальной температуры) φ(x):**



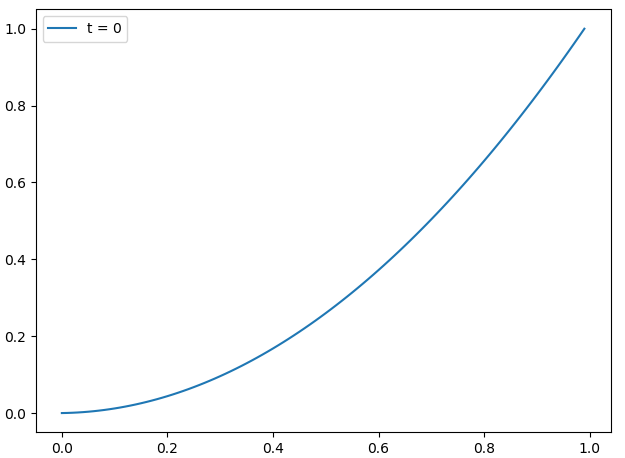
Исходные данные:

**Решение**

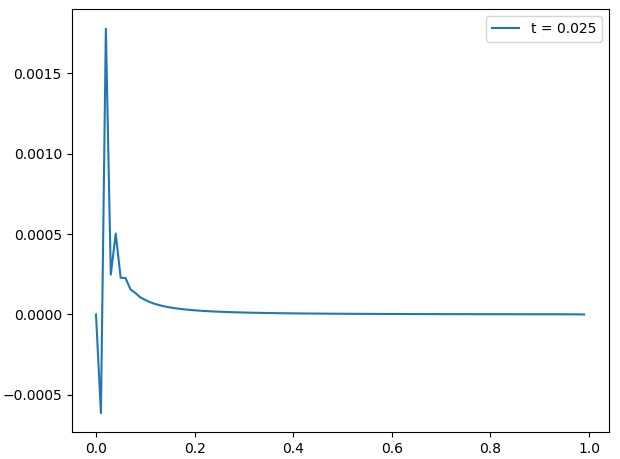
Составим разностную схему:

**Результат работы программы**

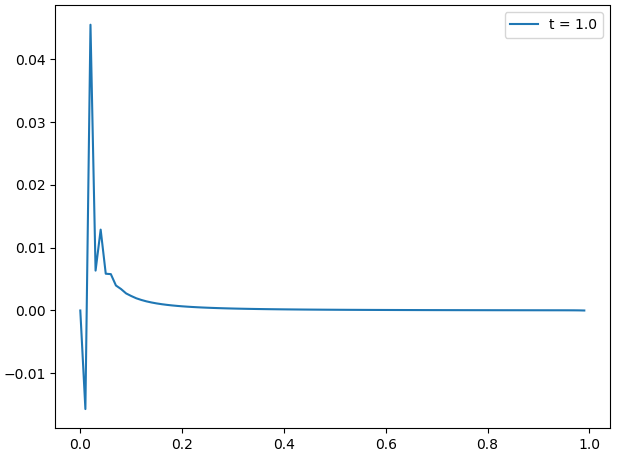
1. **t = 0:**



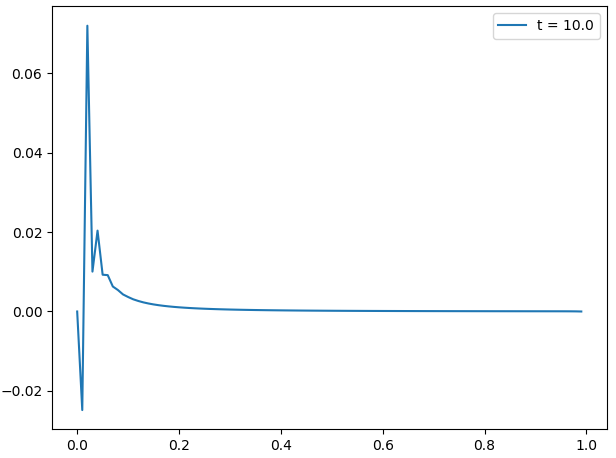
1. **t = :**



1. **t = :**

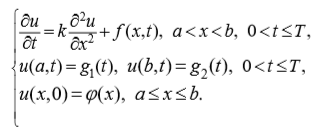


1. **t = :**



**Задание 4**

**Промоделировать нестационарные процессы теплопроводности в зависимости от входных данных задачи. Найти приближенное решение начально-краевой задачи для уравнения теплопроводности:**



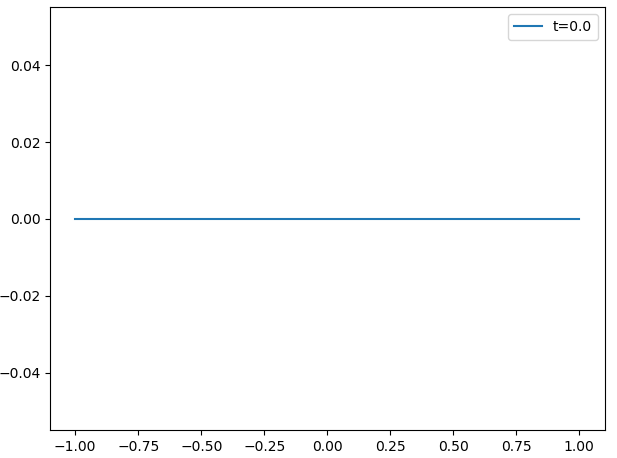
Исходные данные:

**Решение**

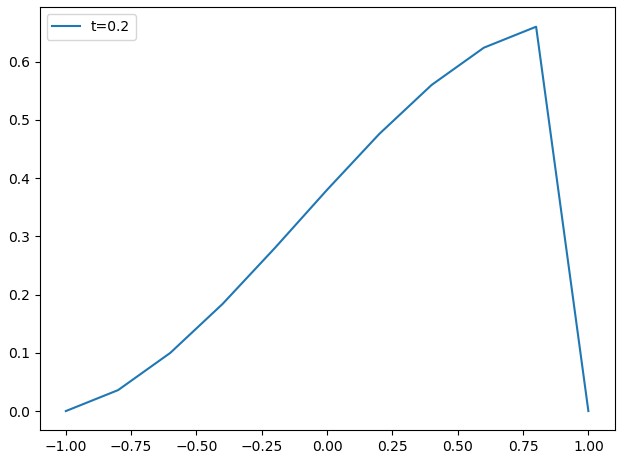
Составим разностную схему:

**Результат работы программы**

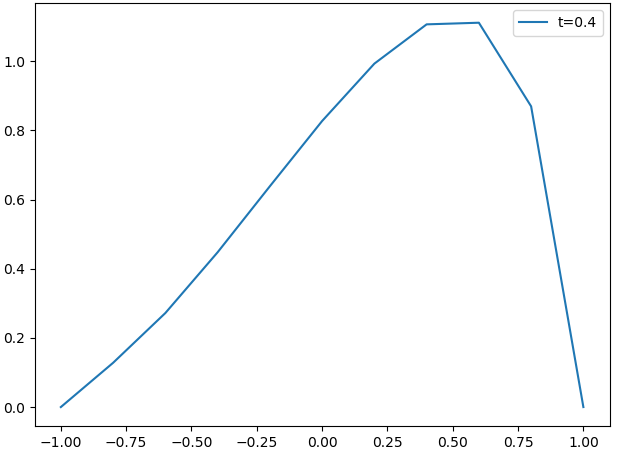
1. **t = 0:**



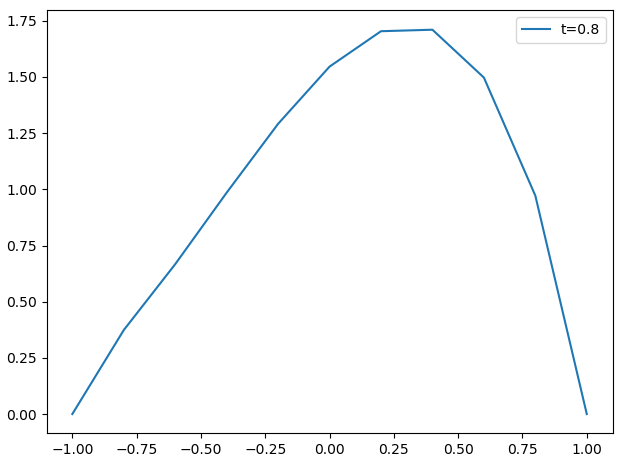
1. **t =**



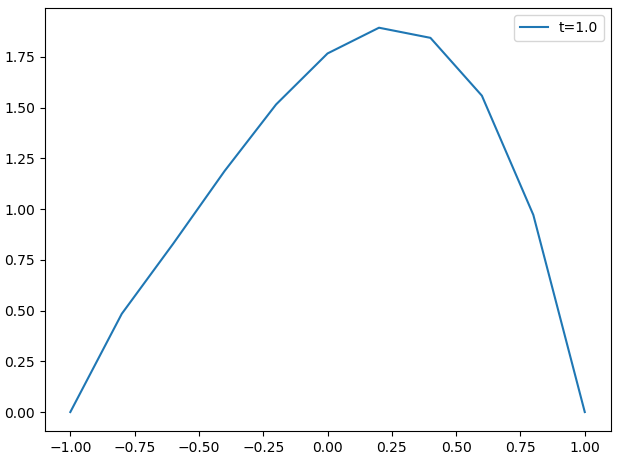
1. **t =**



1. **t =**



1. **t =**



**Код программы**

Задача №1

**import** sympy **as** smp  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
a = 1  
b = 2.2  
x, c1, c2 = smp.symbols(**'x, c1, c2'**)  
f\_x = 6 \* x \*\* 2 - 3 \* x  
  
**def** get\_task1\_coef(coef):  
 ua = coef[**'ua'**]  
 ub = coef[**'ub'**]  
 c = coef[**'c'**]  
 **for** i, ((ua, ub), c\_) **in** enumerate(zip(zip(ua, ub), c)):  
 **if** i **in** [0, 1, 2, 3]:  
 **continue** k\_x = coef[**'k(x)'**] \* c\_  
 df\_x = smp.integrate(f\_x, x)  
 du\_dx = - 1 / k\_x \* df\_x + c1 / k\_x  
 **if** i == 3:  
 du\_dx = - k\_x \* df\_x + c1 \* k\_x  
 u\_x = smp.integrate(du\_dx, x) + c2  
 s1 = u\_x.subs(x, a) - ua  
 s2 = u\_x.subs(x, b) - ub  
 solve = smp.linsolve([s1, s2], c1, c2)  
 c\_1, c\_2 = next(iter(solve))  
 u\_x\_ans = u\_x.subs({c1: c\_1, c2: c\_2})  
 print(u\_x\_ans)  
 plot(u\_x\_ans, i)  
  
**def** plot(u\_x, i):  
 lam\_x = smp.lambdify(x, u\_x, modules=[**'numpy'**])  
 x\_vals = np.arange(a, b + 0.01, 0.01)  
 y\_vals = lam\_x(x\_vals)  
 plt.plot(x\_vals, y\_vals, label=**"Набор: {}"**.format(i+1))  
 plt.ylabel(**"y"**)  
 plt.xlabel(**"x"**)  
  
  
**def** main():  
  
 coef = {**'ua'**: [-2]\*4 + [2, -2, 2], **'ub'**: [2]\*5 + [-2]\*2, **'c'**: [1, 2, 1.5] + [1]\*4, **'k(x)'**: x\*\*(-2)}  
  
 get\_task1\_coef(coef)  
 *# print(c\_1, c\_2)* plt.legend()  
 plt.show()  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

Задача №2

**import** sympy **as** smp  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
a = 1  
b = 2.2  
x, c1, c2 = smp.symbols(**'x, c1, c2'**)  
f\_x = 6 \* x \*\* 2 - 3 \* x  
  
  
**def** solve\_3\_23(coef):  
 ua = coef[**'ua'**]  
 ub = coef[**'ub'**]  
 c = coef[**'c'**]  
 all\_k = coef[**'k(x)'**]  
 x\_vals = get\_x()  
 y\_vals = [0] \* len(x\_vals)  
 df\_x = smp.integrate(f\_x, x)  
 **for** all\_k\_x **in** all\_k:  
 **for** en, i **in** enumerate(x\_vals):  
  
 **if** len(all\_k\_x) == 2:  
 **if** i <= (a + b) / 2:  
 k\_x = all\_k\_x[0] \* c  
 **else**:  
 k\_x = all\_k\_x[1] \* c  
 **else**:  
 **if** i <= a + (b - a) / 3:  
 k\_x = all\_k\_x[0] \* c  
 **elif** i > a + 2 \* (b - a) / 3:  
 k\_x = all\_k\_x[2] \* c  
 **else**:  
 k\_x = all\_k\_x[1] \* c  
  
 du\_dx = - 1 / k\_x \* df\_x + c1 / k\_x  
 u\_x = smp.integrate(du\_dx, x) + c2  
 s1 = u\_x.subs(x, a) - ua  
 s2 = u\_x.subs(x, b) - ub  
 solv = smp.linsolve([s1, s2], c1, c2)  
 c\_1, c\_2 = next(iter(solv))  
 u\_x\_ans = u\_x.subs({c1: c\_1, c2: c\_2})  
 *# print(u\_x\_ans)* y\_vals[en] = u\_x\_ans.subs(x, i)  
 **if** len(all\_k\_x) == 2:  
 plot\_1(x\_vals, y\_vals, all\_k\_x)  
 **else**:  
 plot\_2(x\_vals, y\_vals, all\_k\_x)  
 plt.legend()  
 plt.show()  
  
  
**def** solve\_3\_4(coef):  
 ua = coef[**'ua'**]  
 ub = coef[**'ub'**]  
 c = coef[**'c'**]  
 all\_x0 = coef[**'x0'**]  
 delta\_func = coef[**'delta\_func'**]  
 k\_x = coef[**'k(x)'**]  
  
 **for** x0\_case, c\_\_ **in** zip(all\_x0, c):  
 **for** i, (x0, c\_) **in** enumerate(zip(x0\_case, c\_\_)):  
 func\_x = c\_ \* delta\_func \* (x - x0)  
 df\_x = smp.integrate(func\_x, x)  
 du\_dx = - 1 / k\_x \* df\_x + c1 / k\_x  
 u\_x = smp.integrate(du\_dx, x) + c2  
 s1 = u\_x.subs(x, a) - ua  
 s2 = u\_x.subs(x, b) - ub  
 solve = smp.linsolve([s1, s2], c1, c2)  
 c\_1, c\_2 = next(iter(solve))  
 u\_x\_ans = u\_x.subs({c1: c\_1, c2: c\_2})plot(u\_x\_ans, i, x0, c\_)

plt.legend()  
 plt.show()  
  
  
**def** get\_x():  
 **return** list(np.linspace(a, b, 150))  
  
  
**def** plot\_2(x\_vals, y\_vals, k\_x):  
 plt.plot(x\_vals[:50], y\_vals[:50], label=**"k1: {}"**.format(k\_x[0]))  
 plt.plot(x\_vals[50:100], y\_vals[50:100], label=**"k2: {}"**.format(k\_x[1]))  
 plt.plot(x\_vals[100:], y\_vals[100:], label=**"k3: {}"**.format(k\_x[2]))  
 plt.ylabel(**"y"**)  
 plt.xlabel(**"x"**)  
  
  
**def** plot\_1(x\_vals, y\_vals, k\_x):  
 plt.plot(x\_vals[:75], y\_vals[:75], label=**"k1: {}"**.format(k\_x[0]))  
 plt.plot(x\_vals[75:], y\_vals[75:], label=**"k2: {}"**.format(k\_x[1]))  
 plt.ylabel(**"y"**)  
 plt.xlabel(**"x"**)  
  
  
**def** plot(u\_x, i, x0, c):  
 lam\_x = smp.lambdify(x, u\_x, modules=[**'numpy'**])  
 x\_vals = np.arange(a, b + 0.01, 0.01)  
 y\_vals = lam\_x(x\_vals)  
 plt.ylabel(**"y"**)  
 plt.xlabel(**"x"**)  
  
  
**def** main():  
  
 coef = {**'ua'**: -2, **'ub'**: 2, **'c'**: 1, **'k(x)'**: [[x\*\*3, x\*\*(-3)],  
 [x\*\*(-3), x\*\*3]]}  
  
 coef\_2 = {**'ua'**: -2, **'ub'**: 2, **'c'**: 1, **'k(x)'**: [[1, x\*\*2, x\*\*4],  
 [x\*\*4, x\*\*2, 1],  
 [x \*\* 2, 2\*x\*\*2, x\*\*2],  
 [20 \* x \*\* 3, x\*\*3, 20 \* x\*\*3]]}  
  
 centr\_pos = (a + b) / 2  
 coef\_3 = {**'ua'**: -2, **'ub'**: 2, **'c'**: [[1], [1, 1], [1, 4]], **'delta\_func'**: np.sin(10),  
 **'k(x)'**: x\*\*(-2), **'x0'**: [[centr\_pos],  
 [centr\_pos - 0.2, centr\_pos + 0.2],  
 [centr\_pos - 0.4, centr\_pos + 0.4]]}  
  
 *# solve\_3\_23(coef\_2)* solve\_3\_4(coef\_3)**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

Задача №3

**import** sympy **as** smp  
**import** numpy **as** np  
**from** matplotlib **import** pyplot **as** plt  
  
  
x = smp.symbols(**'x'**)  
h = 0.01  
tau = 0.05  
start = 0  
finish = 1  
  
  
**def** solve(t):  
 Y = []  
 A = []  
 B = []  
 k\_x = x  
 f\_x = 1 / x  
 i = 0  
 **for** xi **in** np.arange(start + h, finish, h):  
 **if** t != 0:  
 a = (2 \* tau \* k\_x - smp.diff(k\_x, x) \* tau \* h + h\*\*2).subs(x, xi)  
 b = (4 \* tau \* k\_x).subs(x, xi)  
 c = (h\*\*2 - smp.diff(k\_x, x) \* tau \* h - 2 \* tau \* k\_x).subs(x, xi)  
 d = (f\_x \* (1 - smp.exp(-t)) \* 2 \* tau \* h\*\*2).subs(x, xi)  
  
 **if** i == 0:  
 Ai = - c / b  
 Bi = d / b  
 **else**:  
 Ai = -c / (b + a \* A[-1])  
 Bi = (d - a \* B[-1]) / (b + a \* A[-1])  
  
 A.append(Ai)  
 B.append(Bi)  
 **else**:  
 Y.append(xi \*\* 2)  
 i += 1  
  
 **if** t == 0:  
 Y.append(finish \*\* 2)  
 **return** Y  
  
 Y = [0] \* (i + 1)  
  
 **while** i > 1:  
 i -= 1  
 Y[i] = A[i] \* Y[i + 1] + B[i]  
  
 **return** Y  
  
  
**def** main():  
 t = [0, 0.5 \* tau, 20 \* tau, 200 \* tau]  
 x\_arr = np.arange(start, finish, h)  
  
 **for** t\_ **in** t:  
 plt.plot(x\_arr, solve(t\_), label=**"t = {}"**.format(t\_))  
 plt.legend()  
 plt.show()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задача №4

**import** sympy **as** smp  
**import** numpy **as** np  
**from** matplotlib **import** pyplot **as** plt  
  
  
x = smp.Symbol(**'x'**)  
start = -1  
finish = 1  
k = 0.2  
T = 1  
  
  
**def** solve(h, t):  
 f\_x = 1 - x\*\*2  
 A = []  
 B = []  
 Y = []  
 i = 0  
 **for** xi **in** np.arange(start, finish, h):  
 **if** t != 0:  
 a = - h\*\*2 - k \* t  
 b = 2 \* t \* k  
 c = h \*\* 2 - k \* t  
 d = (t \* h \*\* 2 \* f\_x).subs(x, xi)  
  
 **if** i == 0:  
 Ai = - c / b  
 Bi = d / b  
 **else**:  
 Ai = -c / (b + a \* A[-1])  
 Bi = (d - a \* B[-1]) / (b + a \* A[-1])  
  
 A.append(Ai)  
 B.append(Bi)  
 **else**:  
 Y.append(0)  
 i += 1  
  
 **if** t == 0:  
 Y.append(0)  
 **return** Y  
  
 Y = [0] \* (i + 1)  
  
 **while** i > 1:  
 i -= 1  
 Y[i] = A[i] \* Y[i + 1] + B[i]  
  
 **return** Y  
  
  
**def** main():  
 y\_arrs = []  
 h = (finish - start) / 10  
 tau = 0.5 \* h \*\* 2 / k  
 t = [0] \* 2000  
 val = tau  
 **for** i **in** range(2000):  
 y\_arrs.append(solve(h, t[i]))  
 *# print(i, t[i])* val = round(2 \* i \* tau, 4)  
 t[i + 1] = val  
  
 **if** t[i + 1] > T:  
 **break** y\_arrs.append(solve(h, T))  
 x\_arr = np.arange(start, finish + h, h)  
 **for** i, y **in** enumerate(y\_arrs):  
 plt.plot(x\_arr, y, label=**"t={}"**.format(round(t[i], 4)))  
 plt.legend()  
 plt.savefig(**'ans-task4/t={}.png'**.format(t[i]))  
 plt.show()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()