Кафедра 204 Вариант 2К (РС-2)

ЗАДАНИЕ

на курсовую работу по курсу «ТЕПЛОПЕРЕДАЧА»

Студент Зюзь Павел Группа М2В 301Б 21 Руководитель – доц. Семенов А.А.

Тема: Расчет нестационарного нагревания стенок неохлаждаемого реактивного сопла.

ОСНОВНЫЕ ИСХОДНЫЕ ДАННЫЕ ДЛЯ РАСЧЕТА:

1. Число М на срезе сопла Мсреза=5
2. Полуугол раствора конического сопла *v*=10 град.
3. Массовый расход через сопло G=5 кг/с.
4. Давление в камере сгорания p\*=1,98×106 Па.
5. Показатель адиабаты для продуктов сгорания k=1,33.
6. Температура торможения T\*=2500 К.
7. Время работы двигателя τ =10 с.
8. Начальная температура стенок сопла Tн=273 К.

ДОПОЛНИТЕЛЬНЫЕ ДАННЫЕ:

Допустимая температура стенки сопла Tдоп и ее материал задаются самостоятельно, причем Tдоп принимается близкой или равной температуре плавления материала.

Теплофизические константы продуктов сгорания условно принимаются такими же, как у CO2 при соответствующих давлениях и температурах.

ЗАДАЧА РАСЧЁТА:

1. Найти распределение толщины стенки по длине сопла δ(x), при котором в конце работы двигателя температура внутренней поверхности сопла нигде не превышала Tдоп.
2. При найденном распределении рассчитать распределение температуры по внешней поверхности сопла к концу работы двигателя.

Дата выдачи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. Дата сдачи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Семенов А.А.

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc152655715)

[1. Газодинамический расчет сопла 5](#_Toc152655716)

[2. Расчет теплообмена между газом и внутренней поверхностью сопла (конвекция) 10](#_Toc152655717)

[3. Тепловой расчет стенки сопла (нестационарная теплопроводность) 19](#_Toc152655718)

[Заключение 25](#_Toc152655719)

[Список литературы 26](#_Toc152655720)

[Приложение. 27](#_Toc152655721)

# Введение

Расчет нестационарного нагревания стенок приводится с помощью программы, написанной на языке программирования python.

Расчет состоит из трех частей.

1. Газодинамический расчет сопла
2. Расчет теплообмена между газом и внутренней поверхностью сопла (конвекция)
3. Тепловой расчет стенки сопла (нестационарная теплопроводность)

Первые два раздела считаются аналитически, по формулам газодинамики и теплопередачи, расчет в программе лишь позволяет сократить время расчета и уменьшить вероятность ошибки.

Третий раздел считается численно (явный метод расчета нестационарной теплопроводности, подробное описание в третьем разделе). Численной метод позволяет точнее рассчитать температуру в стенке и толщину, а также выполнить расчет в большем количестве точек. С другой стороны, время расчета довольно велико (по сравнению с аналитическим расчетом первых двух разделов).

Расчет выполнялся в среде Jupiter notebook, на python версии 3.11.2.

Используемые сторонние библиотеки:

* numpy (версия 1.26.0) для работы с массивами данных
* matplotlib (версия 3.8.0) для отображения графиков

*Текст программы. Импорт библиотек.*

|  |
| --- |
| import numpy as np  import math  import matplotlib.pyplot as plt  import sys |

Специализированные пакеты и библиотеки для расчета теплопередачи и газодинамики не использовались.

Полный текст программы приводится в конец в приложении.

Перед тем как начать расчет определим дополнительные входные данные.

Для сопла выберем вольфрамовый сплав.

число маха на входе

коэффициент динамической вязкости газа

коэффициент теплопроводности газа

плотность материала стенки (вольфрамовый сплав)

коэффициент теплопроводности материала стенки (вольфрамовый сплав)

теплоемкость материала стенки (вольфрамовый сплав)

температура плавления материала стенки (вольфрамовый сплав)

*Текст программы. Ввод начальных данных.*

|  |
| --- |
| M\_sr = 5 # число М на срезе сопла  nu = 10 # Полуугол раствора конического сопла, град.  G = 5 # Массовый расход через сопло, кг/с.  p\_0 = 1.98e6 # Давление в камере сгорания, Па.  k = 1.33 # Показатель адиабаты для продуктов сгорания  T\_0 = 2500 # Температура торможения, К.  t = 10 # Время работы двигателя, с.  T\_n = 273 # Начальная температура стенок сопла, К.  R = 287 # Универсальная газовая постоянная для воздуха  M\_vh = 0.1 # число M на входе.  mu = 20e-6 # Па\*с коэффициент динамической вязкости  lam\_t = 7.696e-2 # Вт\*(м\*К) - коэффициент теплопроводности газа  ro\_m = 11500 # кг/м^3 Плотность материала стенки (вольфрамовый сплав)  lam\_m = 136.5 # Вт/(м\*К) Коэффициент теплопроводности материала стенки (вольфрамовый сплав)  c\_m = 950 # Дж/(кг\*К) теплоемкость материала стенки (вольфрамовый сплав)  T\_pl = 2200 # К - температура плавления материала стенки (вольфрамовый сплав) |

# Газодинамический расчет сопла

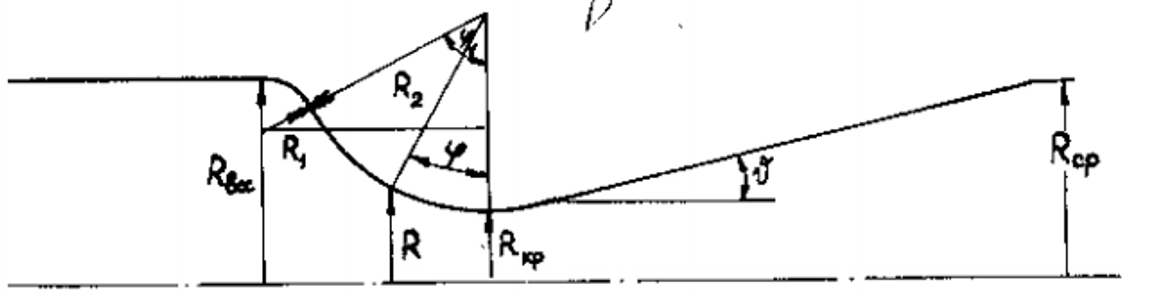


Рис. 1. Конфигурация сопла.

1.1 Рассчитаем площадь критического сечения сопла. Для этого найдем радиусы закругления сопла .

Длина входной части сопла:

Находим площадь и радиус критического сечения.

Для этого запишем уравнение расхода:

, где

Тогда

1.2 Площадь и радиус входного и выходного сечения. Из уравнения газовой функции:

Для этого переводим . Значения и формула перевода

*Текст программы. Расчет критического и выходного сечений.*

|  |
| --- |
| def qlambda(nashe\_lambda,k=1.4):      q\_lambda=nashe\_lambda\*pow(((k+1)/2),(1/(k-1)))\*pow((1-((k-1)/(k+1))\*pow(nashe\_lambda,2)),(1/(k-1)))      return q\_lambda  def lam\_Mlambda(M\_lambda,k):      nashe\_lambda=pow((((k+1)/2)\*pow(M\_lambda,2)/(1+((k-1)/2)\*pow(M\_lambda,2))),0.5)      return nashe\_lambda  def qM(M, k=1.4):      lam = lam\_Mlambda(M, k)      return qlambda(lam, k)  F\_vh = F\_kr/qM(M\_vh)  R\_vh = math.sqrt(F\_vh/math.pi)  F\_sr = F\_kr/qM(M\_sr)  R\_sr = math.sqrt(F\_sr/math.pi) |

1.3 Расчет длины сопла. Длина сопла по образующей:

Где ,

Длина сопла по оси симметрии:

Где

1.4 Рассчитаем длину образующей в дозвуковом участке сопла

Где

*Текст программы. Расчет длины участков сопла.*

|  |
| --- |
| l\_1 = 3\*R\_vh  R\_1 = 0.63\*R\_vh  R\_2 = 3\*R\_vh  cos\_fi\_n = (R\_2 + R\_kr - (R\_vh - R\_1))/(R\_1 + R\_2)  fi\_n = math.acos(cos\_fi\_n)  # fi\_n = math.asin(1-(R\_vh - R\_kr)/(R\_1 + R\_2))  l\_d = R\_1\*fi\_n + R\_2\*fi\_n  l\_s = (R\_sr - R\_kr)/math.sin(math.pi\*nu/180)  l = l\_1 + l\_d + l\_s  l  x\_d = (R\_1 + R\_2)\*math.sin(fi\_n)  x\_s = (R\_sr - R\_kr)/math.tan(math.pi\*nu/180)  x = l\_1 + x\_d + x\_s  def get\_l\_R(R):      cos\_fi = (R\_2 -(R - R\_kr))/R\_2      fi = math.acos(cos\_fi)      l\_d\_r = R\_1\*fi\_n + R\_2\*(fi\_n - fi)      return l\_1 + l\_d\_r |

1.5 Параметры потока на внешней границе пограничного слоя, образующегося на стенках сопла:

В данном пункте газовые функции также взяты из курса МЖГ.

*Текст программы. Расчет газодинамических параметров сопла.*

|  |
| --- |
| def pilambda(nashe\_lambda,k=1.4):      pi\_lambda=pow((1-((k-1)/(k+1))\*pow(nashe\_lambda,2)),(k/(k-1)))      return pi\_lambda  def taulambda(nashe\_lambda,k=1.4):      tau\_lambda=1-((k-1)/(k+1))\*pow(nashe\_lambda,2)      return tau\_lambda  def epslambda(nashe\_lambda,k=1.4):      eps\_lambda=pow((1-((k-1)/(k+1))\*pow(nashe\_lambda,2)),(1/(k-1)))      return eps\_lambda  def get\_p\_M(M):      lam = lam\_Mlambda(M, k)      return p\_0\*pilambda(lam, k)  def get\_T\_M(M):      lam = lam\_Mlambda(M, k)      return T\_0\*taulambda(lam, k)  def get\_ro\_M(M):      lam = lam\_Mlambda(M, k)      return p\_0/(R\*T\_0)\*epslambda(lam, k) |

# Расчет теплообмена между газом и внутренней поверхностью сопла (конвекция)

Из числа Нуссельта можно определить зависимость коэффициента теплоотдачи от x

где

С начала зададим количество точек расчета и разделим всю длину сопла на равные участки. Было выбран количество точек n = 100.

*Текст программы. Выбор количества точек и разделение длины сопла на участки*

|  |
| --- |
| N = 100  # Зададим линейные размеры  l\_0 = 0  T\_dop = T\_pl - 50  # Массив точек расчета  ar\_x = np.linspace(l\_0, l, N) |

Далее определим зависимость радиуса сечения и площади сечения от x.

*Текст программы. Расчет радиуса и площади сечения от x*

|  |
| --- |
| R\_x = []  fi\_n = math.asin(1-(R\_vh - R\_kr)/(R\_1 + R\_2))  for x\_i in ar\_x:      if x\_i < l\_1:          R\_x.append(R\_vh)      elif x\_i < l\_1 + R\_1\*math.cos(fi\_n):          x\_r\_i = x\_i - l\_1          fi = math.acos(x\_r\_i/R\_1)          R\_x.append(R\_vh - R\_1\*(1 - math.sin(fi)))      elif x\_i < l\_1 + x\_d:          x\_r\_i = x\_d - x\_i + l\_1          fi = math.acos(x\_r\_i/R\_2)          R\_x.append(R\_kr + R\_2\*(1 - math.sin(fi)))      else:          R\_x.append(R\_kr + (x\_i - l\_1 - x\_d)\*math.tan(math.pi\*nu/180))  R\_x = np.array(R\_x)  F\_x = np.pi\*R\_x\*\*2 |

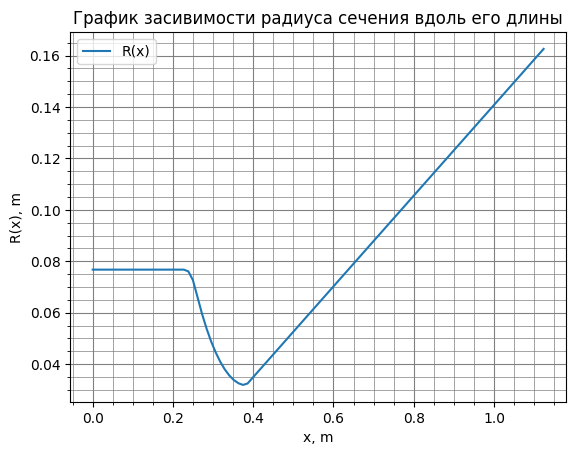


Рис. 2. График зависимости радиуса сечения от x

После этого необходимо посчитать зависимость скорости газа сопла от x.

*Текст программы. Расчет скорости от x*

|  |
| --- |
| def lam\_qlambda(q\_lambda,k):      a=0      b=1      c=2.5      d=0.0000001      epsln=0.00001      pogreshnost=1      nashe\_lambda=a+d      if q\_lambda==0:          q\_lambda==d      while pogreshnost>epsln:          F=((k+1)/(k-1))\*pow(nashe\_lambda,(k-1))-pow(nashe\_lambda,(k+1))-2\*pow(q\_lambda,(k-1))/(k-1)          F\_1=(k+1)\*(pow(nashe\_lambda,(k-2))-pow(nashe\_lambda,k))          F\_2=(k+1)\*(k-2)\*pow(nashe\_lambda,(k-3))-k\*(k+1)\*pow(nashe\_lambda,k-1)          obraz\_x=nashe\_lambda          nashe\_lambda=nashe\_lambda-F/F\_1          pogreshnost=abs(obraz\_x-nashe\_lambda)      rez=[]      rez.append(abs(nashe\_lambda))      nashe\_lambda=b+d      pogreshnost=1      while pogreshnost>epsln:          F=((k+1)/(k-1))\*pow(nashe\_lambda,(k-1))-pow(nashe\_lambda,(k+1))-2\*pow(q\_lambda,(k-1))/(k-1)          F\_1=(k+1)\*(pow(nashe\_lambda,(k-2))-pow(nashe\_lambda,k))          F\_2=(k+1)\*(k-2)\*pow(nashe\_lambda,(k-3))-k\*(k+1)\*pow(nashe\_lambda,k-1)          obraz\_x=nashe\_lambda          nashe\_lambda=nashe\_lambda-F/F\_1          pogreshnost=abs(obraz\_x-nashe\_lambda)      rez.append(abs(nashe\_lambda))      return rez  def Mlambda(nashe\_lambda, k=1.4):      M\_lambda=pow(((2/(k+1))\*pow(nashe\_lambda,2)/(1-((k-1)/(k+1))\*pow(nashe\_lambda,2))),0.5)      return M\_lambda  q\_lambda = F\_kr/F\_x  lam\_x = []  for i, q\_l in enumerate(q\_lambda):      # print(i, q\_l)      lam\_2 = lam\_qlambda(q\_l, k)      if ar\_x[i] < l\_1 + x\_d:          lam\_x.append(lam\_2[0])      else:          lam\_x.append(lam\_2[1])  lam\_x = np.array(lam\_x)  M\_x = Mlambda(lam\_x, k)  T\_x = T\_0\*taulambda(lam\_x)  T\_w = (T\_pl + T\_0)/2  v\_x = M\_x\*np.sqrt(k\*R\*T\_x) |

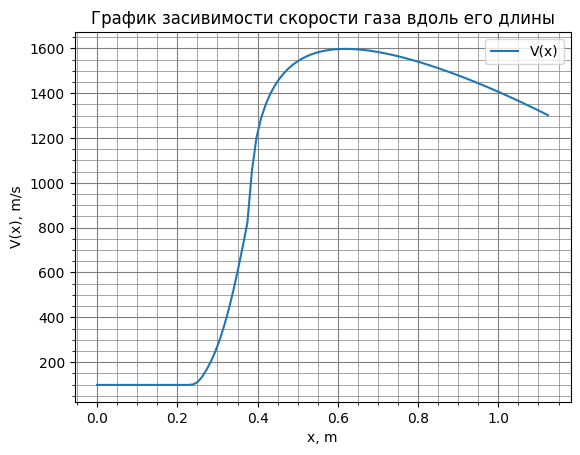


Рис. 3. График зависимости скорости газа от x

После этого посчитаем распределение плотности газа от x.

*Текст программы. Расчет плотности газа от x*

|  |
| --- |
| ro\_x = get\_ro\_M(M\_x) |

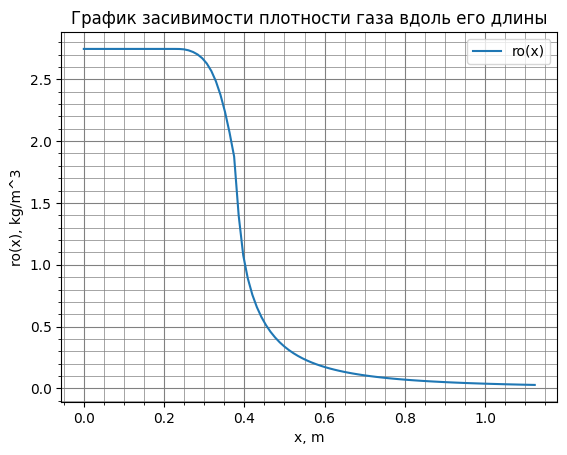


Рис. 4. График зависимости плотности газа от x

Далее найдем эффективную длину сопла в зависимости от x (с учетом поправки на развитие пограничного слоя). Расчет ведется путем численного подсчета определенного интеграла.

*Текст программы. Расчет эффективной длины в зависимости от x*

|  |
| --- |
| def get\_x\_ef(i):      x\_hist = ar\_x[:i+1]      sum\_int = 0      for ii in range(len(x\_hist)-1):            delta\_x = x\_hist[ii+1] - x\_hist[ii]          ff\_i = ro\_x[ii]\*v\_x[ii]\*R\_x[ii]\*\*(5/4)          ff\_ip1 = ro\_x[ii+1]\*v\_x[ii+1]\*R\_x[ii+1]\*\*(5/4)          ff = (ff\_i + ff\_ip1)/2          sum\_int += ff\*delta\_x      res = sum\_int/(ro\_x[i]\*v\_x[i]\*R\_x[i]\*\*(5/4))      if not res:          res = ar\_x[i]      return res  x\_ef = []  for i in range(N):      x\_ef.append(get\_x\_ef(i))  x\_ef = np.array(x\_ef)  x\_ef[0] = 1e-9 |

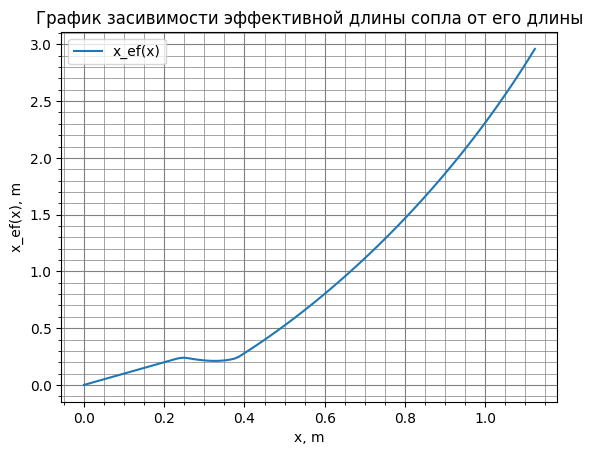


Рис. 5. График зависимости эффективной длины от x

Затем посчитаем коэффициент подобия Re в зависимости от x.

*Текст программы. Расчет Re(x)*

|  |
| --- |
| Re\_x = ro\_x\*v\_x\*x\_ef/mu |

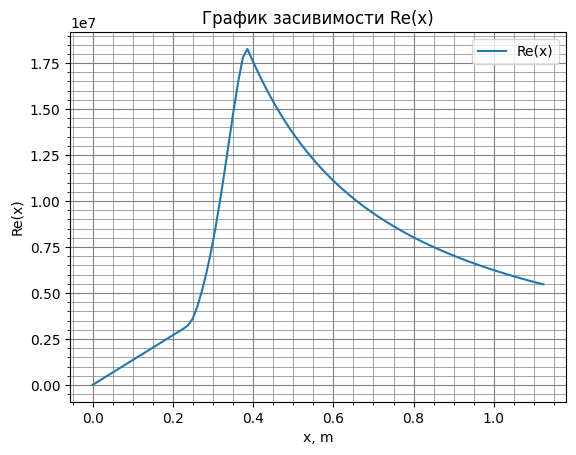


Рис. 6. График зависимости Re(x)

Также рассчитаем коэффициент подобия Pr. Он не зависит от x.

*Текст программы. Расчет Pr*

|  |
| --- |
| c\_p = 1e3 # Дж/(кг\*К), для воздуха  Pr\_x = mu\*c\_p/lam\_t |

Pr = 0.2598752598752599

После этого считаем температуру стенки.

*Текст программы. Расчет температуры стенки в зависимости от x.*

|  |
| --- |
| r\_v = 0.89  T\_e\_x = T\_x\*(1+((k-1)/2)\*r\_v\*M\_x\*\*2) |

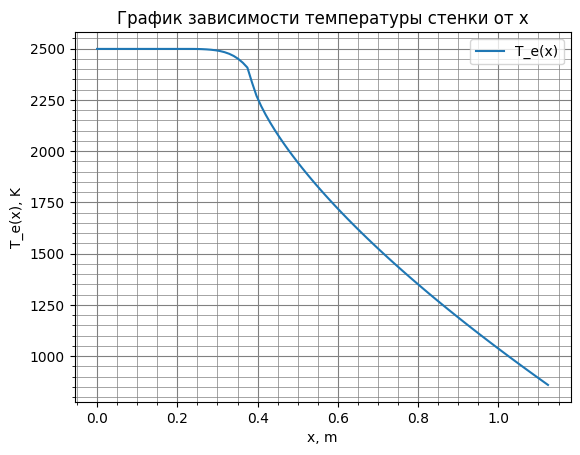


Рис. 7. График температуры стенки от x

Наконец можно рассчитать коэффициент теплоотдачи в зависимости от x.

*Текст программы. Расчет коэффициента теплоотдачи от x.*

|  |
| --- |
| alpha\_x = 0.0296\*lam\_t\*Re\_x\*\*0.8\*Pr\_x\*\*0.43\*(T\_w/T\_e\_x)\*\*0.4\*(1+((k-1)/2)\*r\_v\*M\_x\*\*2)\*\*0.11/x\_ef  alpha\_x[0] = alpha\_x[1] |

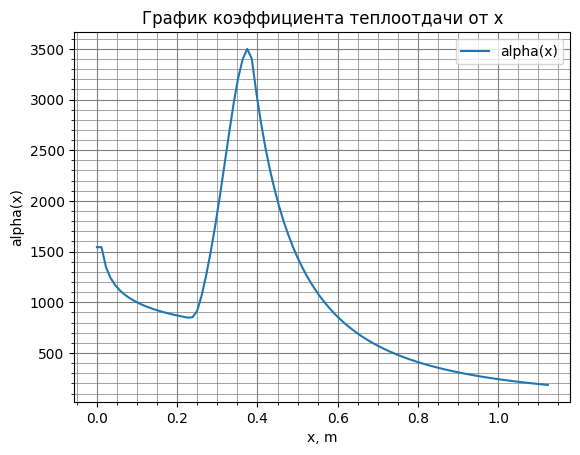


Рис. 8. График зависимости коэффициента теплоотдачи от x

# Тепловой расчет стенки сопла (нестационарная теплопроводность)

Рассчитаем распределение температуры в стенке сопла. Расчет будет выполнен численно, явным методом. По сравнению с методом, использующим номограмму, численный метод позволяет более точно рассчитать толщину стенки. Также численный метод позволяет получить распределение температуры по толщине стенки для любого ее участка в любой момент времени. Недостатком численного расчета является большое время выполнения расчета. Причина этого – расчет делается для каждого участка по x для нескольких толщин, толщина при этом при на каждой итерации подбора увеличивается, пока наибольшая температура стенки не окажется больше максимально допустимой. В итоге требуется достаточно большое количество расчетов. Вторая причина – для расчета используется явный численный метод, у которого есть ограничение по шагу времени, шаг времени должен быть достаточно малым для того, чтобы решение было устойчивым. Далее это будет описано.

Рассчитаем распределение температуры на каждом участке стенки, для нескольких толщин, в результате расчета подберем оптимальную толщину, когда максимальное значение температуры стенки равно или немного меньше максимально допустимой температуры.

Явный численный метод. Стенку делим на промежутки

Для внутреннего слоя k на шаге по времени n + 1

Для крайнего по стенке слоя на шаге по времени n + 1

Условия устойчивости решения:

*Текст программы. Функция для расчета i-го элемента сопла по длине.*

|  |
| --- |
| def calculate\_temp\_x\_step(ind\_x, d, n\_t):        n\_d = 10      delta\_d =  2\*d/n\_d      # n\_t = 5000 # Шаг подбирается исходя из F\_0 <= 0.5      delta\_t = t/n\_t      F\_0 = lam\_m\*delta\_t/(delta\_d\*\*2\*c\_m\*ro\_m)      Bi = alpha\_x[ind\_x]\*delta\_d/lam\_m      assert F\_0 <= 0.5, 'F\_0 = {} > 0.5'.format(F\_0)      assert 1 - 2\*F\_0 - 2\*F\_0\*Bi > 0, '1 - 2\*F\_0 - 2\*F\_0\*Bi = {} <= 0'.format(1 - 2\*F\_0 - 2\*F\_0\*Bi)      # print(d, F\_0, 1 - 2\*F\_0 - 2\*F\_0\*Bi)      Bi = alpha\_x[ind\_x]\*delta\_d/lam\_m      T\_t\_x = np.zeros((n\_t, n\_d))      T\_t\_x[0] = T\_n      # T\_t\_x[0, 0] = 2\*F\_0\*(T\_n + Bi\*T\_e\_x[ind\_x]) + (1 - 2\*F\_0 - 2\*F\_0\*Bi)\*T\_n      # T\_t\_x[0, n\_d-1] = 2\*F\_0\*(T\_n + Bi\*T\_e\_x[ind\_x]) + (1 - 2\*F\_0 - 2\*F\_0\*Bi)\*T\_n      for c\_ind\_t in range(1, n\_t):          T\_t\_x[c\_ind\_t, 0] = 2\*F\_0\*(T\_t\_x[c\_ind\_t-1, 1] + Bi\*T\_e\_x[ind\_x]) + (1 - 2\*F\_0 - 2\*F\_0\*Bi)\*T\_t\_x[c\_ind\_t-1, 0]          T\_t\_x[c\_ind\_t, n\_d-1] = 2\*F\_0\*(T\_t\_x[c\_ind\_t-1, n\_d-2] + Bi\*T\_e\_x[ind\_x]) + (1 - 2\*F\_0 - 2\*F\_0\*Bi)\*T\_t\_x[c\_ind\_t-1, n\_d-1]          for c\_ind\_x in range(1, n\_d-1):              T\_t\_x[c\_ind\_t, c\_ind\_x] = F\_0\*(T\_t\_x[c\_ind\_t-1, c\_ind\_x+1] + T\_t\_x[c\_ind\_t-1, c\_ind\_x-1]) + (1-2\*F\_0)\*T\_t\_x[c\_ind\_t-1, c\_ind\_x]      return T\_t\_x |

*Текст программы. Расчет распределения толщины и температуры внутри стенки сопла.*

|  |
| --- |
| d\_0 = 0.0017  d\_d\_s = 0.0001  n\_t\_0 = 10000  T\_max\_x = []  T\_in\_x = []  d\_x = []  d = d\_0  R\_prev = 0  d\_prev = 0  print('0 % [>' + ' '\*9 + ']', end = '\r')  sys.stdout.flush()  for i in range(len(ar\_x)):      print('{} % ['.format(i+1) + '-'\*(int(i/10)) + '>' + ' '\*(9 - int(i/10)) + ']', end = '\r')      sys.stdout.flush()      T\_max = 0      T\_max\_prev = 0      T\_in\_prev = 0      T\_in = 0      if d\_prev:          d = d\_prev      first\_it = True      increase = None      while True:          d = round(d, 4)          if d == 0:              break          elif first\_it and d == 0.0001:              d = 0.0002          elif d == d\_prev:              d += d\_d\_s          n\_t = n\_t\_0 + int(800\*(d\_0/d)\*\*2.5)          T\_max\_prev = T\_max          T\_in\_prev = T\_in          T\_t\_x = calculate\_temp\_x\_step(i, d, n\_t)          T\_max = T\_t\_x[-1].max()          d\_prev = d          T\_in = T\_t\_x[-1, 4]          # print(i, d, T\_max, T\_in, n\_t)          if first\_it:             increase = T\_max >= T\_dop          else:              if increase and T\_max < T\_dop:                  break              elif not increase and T\_max >= T\_dop:                  break          if increase:              d += d\_d\_s          else:              d -= d\_d\_s          first\_it = False      # print(min(T\_max, T\_max\_prev), min(T\_in, T\_in\_prev))      T\_max\_x.append(min(T\_max, T\_max\_prev))      T\_in\_x.append(min(T\_in, T\_in\_prev))      d\_x.append(d if increase else d\_prev)  d\_x = np.array(d\_x)  T\_in\_x = np.array(T\_in\_x) |

В результате получаем распределение толщины стенки сопла и распределение температуры стенки сопла.

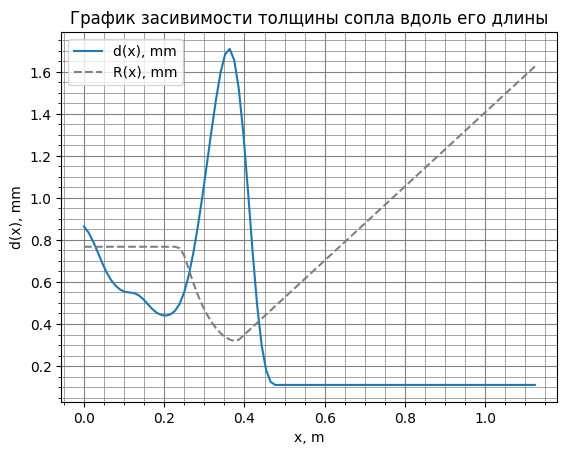


Рис. 9. График распределения толщины стенки сопла от x

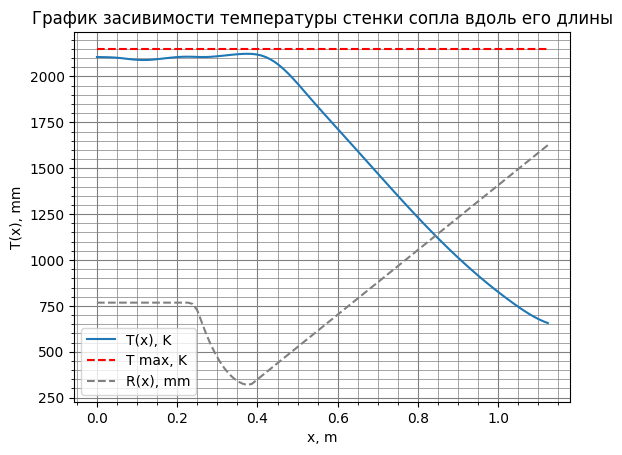


Рис. 10. График распределения температуры внутри стенки сопла от x

# Заключение

Расчет выполнен комбинацией аналитических и численных методов. К преимуществам первых относится возможность получения зависимости в явном виде, доступном для анализа, а также зачастую малое время расчета. Преимущество численных методов в их универсальности, они подходят для тех случаев, для которых недоступны аналитические методы. Преимущество численных методов перед графическими в том, что численные расчеты удобно программировать на ЭВМ, а графические расчеты нет.

Расчет на ЭВМ имеет преимущество над ручным расчетом в том, что путем изменения параметров и затем перезапуска программы мы можем не тратя много времени рассчитать искомые величины.

В нашем случае расчет был выполнен на ноутбуке с процессором intel i7 12700H, 32 Gb оперативной памяти, SSD. Расчет занял не более 8 мин. Точность подбора толщины при этом – 0.1 мм. Точность по оси x – 1/100 длины сопла. Точность расчета всегда можно повысить, увеличив при этом его время.

# Список литературы

1. Исаченко В.П., Осипова В.А., Сукомел А.С.. Теплопередача. Учебник для вузов – изд. четвёртое, переработанное и дополненное. – М.: Энергоиздат, 1981
2. Михеев М.А., Михеева И.М. Основы теплопередачи. Изд. второе. – М.: Энергия, 1977
3. Белов В.П. Сопловые блоки ракетных двигателей. С.-Петербург: Типография БГТУ, 2019

# Приложение.

*Текст программы.*

|  |
| --- |
| import numpy as np  import math  import matplotlib.pyplot as plt  import sys  M\_sr = 5 # число М на срезе сопла  nu = 10 # Полуугол раствора конического сопла, град.  G = 5 # Массовый расход через сопло, кг/с.  p\_0 = 1.98e6 # Давление в камере сгорания, Па.  k = 1.33 # Показатель адиабаты для продуктов сгорания  T\_0 = 2500 # Температура торможения, К.  t = 10 # Время работы двигателя, с.  T\_n = 273 # Начальная температура стенок сопла, К.  R = 287 # Универсальная газовая постоянная для воздуха  M\_vh = 0.1 # Число M на входе.  mu = 20e-6 # Па\*с коэффициент динамической вязкости  lam\_t = 7.696e-2 # Вт\*(м\*К) - коэффициент теплопроводности газа  ro\_m = 11500 # кг/м^3 Плотность материала стенки (вольфрамовый сплав)  lam\_m = 136.5 # Вт/(м\*К) Коэффициент теплопроводности материала стенки (вольфрамовый сплав)  c\_m = 950 # Дж/(кг\*К) теплоемкость материала стенки (вольфрамовый сплав)  T\_pl = 2200 # К - температура плавления материала стенки (вольфрамовый сплав)  m = math.sqrt((2/(k+1))\*\*((k+1)/(k-1))\*(k/R))  F\_kr = G\*math.sqrt(T\_0)/(m\*p\_0)  R\_kr = math.sqrt(F\_kr/math.pi)  def qlambda(nashe\_lambda,k=1.4):      q\_lambda=nashe\_lambda\*pow(((k+1)/2),(1/(k-1)))\*pow((1-((k-1)/(k+1))\*pow(nashe\_lambda,2)),(1/(k-1)))      return q\_lambda  def lam\_Mlambda(M\_lambda,k):      nashe\_lambda=pow((((k+1)/2)\*pow(M\_lambda,2)/(1+((k-1)/2)\*pow(M\_lambda,2))),0.5)      return nashe\_lambda  def qM(M, k=1.4):      lam = lam\_Mlambda(M, k)      return qlambda(lam, k)  F\_vh = F\_kr/qM(M\_vh)  R\_vh = math.sqrt(F\_vh/math.pi)  F\_sr = F\_kr/qM(M\_sr)  R\_sr = math.sqrt(F\_sr/math.pi)  l\_1 = 3\*R\_vh  R\_1 = 0.63\*R\_vh  R\_2 = 3\*R\_vh  cos\_fi\_n = (R\_2 + R\_kr - (R\_vh - R\_1))/(R\_1 + R\_2)  fi\_n = math.acos(cos\_fi\_n)  # fi\_n = math.asin(1-(R\_vh - R\_kr)/(R\_1 + R\_2))  l\_d = R\_1\*fi\_n + R\_2\*fi\_n  l\_s = (R\_sr - R\_kr)/math.sin(math.pi\*nu/180)  l = l\_1 + l\_d + l\_s  l  x\_d = (R\_1 + R\_2)\*math.sin(fi\_n)  x\_s = (R\_sr - R\_kr)/math.tan(math.pi\*nu/180)  x = l\_1 + x\_d + x\_s  def get\_l\_R(R):      cos\_fi = (R\_2 -(R - R\_kr))/R\_2      fi = math.acos(cos\_fi)      l\_d\_r = R\_1\*fi\_n + R\_2\*(fi\_n - fi)      return l\_1 + l\_d\_r  def pilambda(nashe\_lambda,k=1.4):      pi\_lambda=pow((1-((k-1)/(k+1))\*pow(nashe\_lambda,2)),(k/(k-1)))      return pi\_lambda  def taulambda(nashe\_lambda,k=1.4):      tau\_lambda=1-((k-1)/(k+1))\*pow(nashe\_lambda,2)      return tau\_lambda  def epslambda(nashe\_lambda,k=1.4):      eps\_lambda=pow((1-((k-1)/(k+1))\*pow(nashe\_lambda,2)),(1/(k-1)))      return eps\_lambda  def get\_p\_M(M):      lam = lam\_Mlambda(M, k)      return p\_0\*pilambda(lam, k)  def get\_T\_M(M):      lam = lam\_Mlambda(M, k)      return T\_0\*taulambda(lam, k)  def get\_ro\_M(M):      lam = lam\_Mlambda(M, k)      return p\_0/(R\*T\_0)\*epslambda(lam, k)  # Зададим количество точек расчета  N = 100  # Зададим линейные размеры  l\_0 = 0  T\_dop = T\_pl - 50  # Массив точек расчета  ar\_x = np.linspace(l\_0, l, N)  R\_x = []  fi\_n = math.asin(1-(R\_vh - R\_kr)/(R\_1 + R\_2))  for x\_i in ar\_x:      if x\_i < l\_1:          pass          R\_x.append(R\_vh)      elif x\_i < l\_1 + R\_1\*math.cos(fi\_n):          x\_r\_i = x\_i - l\_1          fi = math.acos(x\_r\_i/R\_1)          R\_x.append(R\_vh - R\_1\*(1 - math.sin(fi)))      elif x\_i < l\_1 + x\_d:          x\_r\_i = x\_d - x\_i + l\_1          fi = math.acos(x\_r\_i/R\_2)          R\_x.append(R\_kr + R\_2\*(1 - math.sin(fi)))      else:          R\_x.append(R\_kr + (x\_i - l\_1 - x\_d)\*math.tan(math.pi\*nu/180))  R\_x = np.array(R\_x)  plt.plot(ar\_x, R\_x, label='R(x)')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('R(x), m')  plt.legend()  plt.minorticks\_on()  plt.title('График засивимости радиуса сечения вдоль его длины')  plt.show()  F\_x = np.pi\*R\_x\*\*2  def lam\_qlambda(q\_lambda,k):      a=0      b=1      c=2.5      d=0.0000001      epsln=0.00001      pogreshnost=1      nashe\_lambda=a+d      if q\_lambda==0:          q\_lambda==d      while pogreshnost>epsln:          F=((k+1)/(k-1))\*pow(nashe\_lambda,(k-1))-pow(nashe\_lambda,(k+1))-2\*pow(q\_lambda,(k-1))/(k-1)          F\_1=(k+1)\*(pow(nashe\_lambda,(k-2))-pow(nashe\_lambda,k))          F\_2=(k+1)\*(k-2)\*pow(nashe\_lambda,(k-3))-k\*(k+1)\*pow(nashe\_lambda,k-1)          obraz\_x=nashe\_lambda          nashe\_lambda=nashe\_lambda-F/F\_1          pogreshnost=abs(obraz\_x-nashe\_lambda)      rez=[]      rez.append(abs(nashe\_lambda))      nashe\_lambda=b+d      pogreshnost=1      while pogreshnost>epsln:          F=((k+1)/(k-1))\*pow(nashe\_lambda,(k-1))-pow(nashe\_lambda,(k+1))-2\*pow(q\_lambda,(k-1))/(k-1)          F\_1=(k+1)\*(pow(nashe\_lambda,(k-2))-pow(nashe\_lambda,k))          F\_2=(k+1)\*(k-2)\*pow(nashe\_lambda,(k-3))-k\*(k+1)\*pow(nashe\_lambda,k-1)          obraz\_x=nashe\_lambda          nashe\_lambda=nashe\_lambda-F/F\_1          pogreshnost=abs(obraz\_x-nashe\_lambda)      rez.append(abs(nashe\_lambda))      return rez  def Mlambda(nashe\_lambda, k=1.4):      M\_lambda=pow(((2/(k+1))\*pow(nashe\_lambda,2)/(1-((k-1)/(k+1))\*pow(nashe\_lambda,2))),0.5)      return M\_lambda  q\_lambda = F\_kr/F\_x  lam\_x = []  for i, q\_l in enumerate(q\_lambda):      # print(i, q\_l)      lam\_2 = lam\_qlambda(q\_l, k)      if ar\_x[i] < l\_1 + x\_d:          lam\_x.append(lam\_2[0])      else:          lam\_x.append(lam\_2[1])  lam\_x = np.array(lam\_x)  M\_x = Mlambda(lam\_x, k)  T\_x = T\_0\*taulambda(lam\_x)  T\_w = (T\_pl + T\_0)/2  v\_x = M\_x\*np.sqrt(k\*R\*T\_x)  plt.plot(ar\_x, v\_x, label='V(x)')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('V(x), m/s')  plt.legend()  plt.minorticks\_on()  plt.title('График засивимости скорости газа вдоль его длины')  plt.show()  ro\_x = get\_ro\_M(M\_x)  plt.plot(ar\_x, ro\_x, label='ro(x)')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('ro(x), kg/m^3')  plt.legend()  plt.minorticks\_on()  plt.title('График засивимости плотности газа вдоль его длины')  plt.show()  def get\_x\_ef(i):      x\_hist = ar\_x[:i+1]      sum\_int = 0      for ii in range(len(x\_hist)-1):            delta\_x = x\_hist[ii+1] - x\_hist[ii]          ff\_i = ro\_x[ii]\*v\_x[ii]\*R\_x[ii]\*\*(5/4)          ff\_ip1 = ro\_x[ii+1]\*v\_x[ii+1]\*R\_x[ii+1]\*\*(5/4)          ff = (ff\_i + ff\_ip1)/2          sum\_int += ff\*delta\_x      res = sum\_int/(ro\_x[i]\*v\_x[i]\*R\_x[i]\*\*(5/4))      if not res:          res = ar\_x[i]      return res  x\_ef = []  for i in range(N):      x\_ef.append(get\_x\_ef(i))  x\_ef = np.array(x\_ef)  x\_ef[0] = 1e-9  plt.plot(ar\_x, x\_ef, label='x\_ef(x)')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('x\_ef(x), m')  plt.legend()  plt.minorticks\_on()  plt.title('График засивимости эффективной длины сопла от его длины')  plt.show()  Re\_x = ro\_x\*v\_x\*x\_ef/mu  plt.plot(ar\_x, Re\_x, label='Re(x)')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('Re(x)')  plt.legend()  plt.minorticks\_on()  plt.title('График засивимости Re(x)')  plt.show()  c\_p = 1e3 # Дж/(кг\*К), для воздуха  Pr\_x = mu\*c\_p/lam\_t  Pr\_x  r\_v = 0.89  T\_e\_x = T\_x\*(1+((k-1)/2)\*r\_v\*M\_x\*\*2)  plt.plot(ar\_x, T\_e\_x, label='T\_e(x)')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('T\_e(x), K')  plt.legend()  plt.minorticks\_on()  plt.title('График зависимости температуры стенки от x')  plt.show()  alpha\_x = 0.0296\*lam\_t\*Re\_x\*\*0.8\*Pr\_x\*\*0.43\*(T\_w/T\_e\_x)\*\*0.4\*(1+((k-1)/2)\*r\_v\*M\_x\*\*2)\*\*0.11/x\_ef  alpha\_x[0] = alpha\_x[1]  plt.plot(ar\_x, alpha\_x, label='alpha(x)')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('alpha(x)')  plt.legend()  plt.minorticks\_on()  plt.title('График коэффициента теплоотдачи от x')  plt.show()  def calculate\_temp\_x\_step(ind\_x, d, n\_t):        n\_d = 10      delta\_d =  2\*d/n\_d      # n\_t = 5000 # Шаг подбирается исходя из F\_0 <= 0.5      delta\_t = t/n\_t      F\_0 = lam\_m\*delta\_t/(delta\_d\*\*2\*c\_m\*ro\_m)      Bi = alpha\_x[ind\_x]\*delta\_d/lam\_m      assert F\_0 <= 0.5, 'F\_0 = {} > 0.5'.format(F\_0)      assert 1 - 2\*F\_0 - 2\*F\_0\*Bi > 0, '1 - 2\*F\_0 - 2\*F\_0\*Bi = {} <= 0'.format(1 - 2\*F\_0 - 2\*F\_0\*Bi)      # print(d, F\_0, 1 - 2\*F\_0 - 2\*F\_0\*Bi)      Bi = alpha\_x[ind\_x]\*delta\_d/lam\_m      T\_t\_x = np.zeros((n\_t, n\_d))      T\_t\_x[0] = T\_n      # T\_t\_x[0, 0] = 2\*F\_0\*(T\_n + Bi\*T\_e\_x[ind\_x]) + (1 - 2\*F\_0 - 2\*F\_0\*Bi)\*T\_n      # T\_t\_x[0, n\_d-1] = 2\*F\_0\*(T\_n + Bi\*T\_e\_x[ind\_x]) + (1 - 2\*F\_0 - 2\*F\_0\*Bi)\*T\_n      for c\_ind\_t in range(1, n\_t):          T\_t\_x[c\_ind\_t, 0] = 2\*F\_0\*(T\_t\_x[c\_ind\_t-1, 1] + Bi\*T\_e\_x[ind\_x]) + (1 - 2\*F\_0 - 2\*F\_0\*Bi)\*T\_t\_x[c\_ind\_t-1, 0]          T\_t\_x[c\_ind\_t, n\_d-1] = 2\*F\_0\*(T\_t\_x[c\_ind\_t-1, n\_d-2] + Bi\*T\_e\_x[ind\_x]) + (1 - 2\*F\_0 - 2\*F\_0\*Bi)\*T\_t\_x[c\_ind\_t-1, n\_d-1]          for c\_ind\_x in range(1, n\_d-1):              T\_t\_x[c\_ind\_t, c\_ind\_x] = F\_0\*(T\_t\_x[c\_ind\_t-1, c\_ind\_x+1] + T\_t\_x[c\_ind\_t-1, c\_ind\_x-1]) + (1-2\*F\_0)\*T\_t\_x[c\_ind\_t-1, c\_ind\_x]      return T\_t\_x  d\_0 = 0.0017  d\_d\_s = 0.0001  n\_t\_0 = 10000  T\_max\_x = []  T\_in\_x = []  d\_x = []  d = d\_0  R\_prev = 0  d\_prev = 0  print('0 % [>' + ' '\*9 + ']', end = '\r')  sys.stdout.flush()  for i in range(len(ar\_x)):      print('{} % ['.format(i+1) + '-'\*(int(i/10)) + '>' + ' '\*(9 - int(i/10)) + ']', end = '\r')      sys.stdout.flush()      T\_max = 0      T\_max\_prev = 0      T\_in\_prev = 0      T\_in = 0      if d\_prev:          d = d\_prev      first\_it = True      increase = None      while True:          d = round(d, 4)          if d == 0:              break          elif first\_it and d == 0.0001:              d = 0.0002          elif d == d\_prev:              d += d\_d\_s          n\_t = n\_t\_0 + int(800\*(d\_0/d)\*\*2.5)          T\_max\_prev = T\_max          T\_in\_prev = T\_in          T\_t\_x = calculate\_temp\_x\_step(i, d, n\_t)          T\_max = T\_t\_x[-1].max()          d\_prev = d          T\_in = T\_t\_x[-1, 4]          # print(i, d, T\_max, T\_in, n\_t)          if first\_it:             increase = T\_max >= T\_dop          else:              if increase and T\_max < T\_dop:                  break              elif not increase and T\_max >= T\_dop:                  break          if increase:              d += d\_d\_s          else:              d -= d\_d\_s          first\_it = False      # print(min(T\_max, T\_max\_prev), min(T\_in, T\_in\_prev))      T\_max\_x.append(min(T\_max, T\_max\_prev))      T\_in\_x.append(min(T\_in, T\_in\_prev))      d\_x.append(d if increase else d\_prev)  d\_x = np.array(d\_x)  T\_in\_x = np.array(T\_in\_x)  ar\_x\_add = [ar\_x[0] - (ar\_x[1] - ar\_x[0])]  d\_x\_add = [d\_x[0]]  for i in range(19):      ar\_x\_add.append(ar\_x\_add[-1] - (ar\_x[1] - ar\_x[0]))      d\_x\_add.append(d\_x[0])  ar\_x\_add\_2 = [ar\_x[-1] + (ar\_x[1] - ar\_x[0])]  d\_x\_add\_2 = [d\_x[-1]]  for i in range(19):      ar\_x\_add\_2.append(ar\_x\_add\_2[-1] + (ar\_x[1] - ar\_x[0]))      d\_x\_add\_2.append(d\_x[-1])  ar\_x\_new = np.array(ar\_x\_add + list(ar\_x) + ar\_x\_add\_2)  d\_x\_new  = np.array(d\_x\_add + list(d\_x) + d\_x\_add\_2)  ii = ar\_x\_new.argsort()  ar\_x\_new = ar\_x\_new[ii]  d\_x\_new = d\_x\_new[ii]  w=np.hanning(10)  d\_x\_smooth=np.convolve(w/w.sum(), d\_x\_new, mode='same')  d\_x\_smooth = d\_x\_smooth[20:N+20]\*1.1  plt.plot(ar\_x, d\_x\_smooth\*1000, label='d(x), mm')  plt.plot(ar\_x, R\_x\*10, label='R(x), mm', color='gray', linestyle='--')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('d(x), mm')  plt.legend()  plt.minorticks\_on()  plt.title('График засивимости толщины сопла вдоль его длины')  plt.show()  w=np.hanning(20)  T\_in\_x\_add = 20\*[T\_in\_x[0]]  T\_in\_x\_add\_2 = 20\*[T\_in\_x[-1]]  T\_in\_x\_new  = np.array(T\_in\_x\_add + list(T\_in\_x) + T\_in\_x\_add\_2)  T\_in\_x\_smooth = np.convolve(w/w.sum(), T\_in\_x\_new, mode='same')  T\_in\_x\_smooth = T\_in\_x\_smooth[20:N+20]  T\_dop\_x = np.ones\_like(T\_in\_x)\*T\_dop  plt.plot(ar\_x, T\_in\_x\_smooth, label='T(x), K')  plt.plot(ar\_x, T\_dop\_x, label='T max, K', color='red', linestyle='--')  plt.plot(ar\_x, R\_x\*10000, label='R(x), mm', color='gray', linestyle='--')  plt.grid(True, which='major', linewidth=0.8, color='gray')  plt.grid(True, which='minor', linewidth=0.5, color='gray')  plt.xlabel('x, m')  plt.ylabel('T(x), mm')  plt.legend()  plt.minorticks\_on()  plt.title('График зависимости температуры внутри стенки сопла вдоль его длины')  plt.show() |