ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«БАШКИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**КАФЕДРА МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ**

РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА

по дисциплине «Непрерывные математические модели»

Выполнили:

Студент(ка): 1 курса очной формы обучения

01.04.02 «Прикладная математика и информатика»

Математическое моделирование

Абрамов Егор Александрович

Руководитель

к.ф.-м.н., доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.М. Ефимов

(подпись)

Оценка при защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

УФА – 2021

**Оптимизация вертикального подъёма ракеты**

**Введение**

Методы нелинейной оптимизации широко применяются при проектировании машин и механизмов. Указанные методы применяются и в ракетостроении, например, для оптимизации вертикального подъёма ракеты. Одной из наиболее известных задач в этой области является задача Годдарда.

Задача Годдарда рассматривается во многих работах по теории управления. В классической постановке она заключается в максимизации высоты вертикального подъема ракеты при наличии ограничений на расход топлива. Известно, что особенностью оптимальных траекторий в данной задаче является наличие особых участков, которые, однако, обычно могут быть обнаружены лишь с использованием специальных численных методов. Тем более затруднен их поиск в случае, когда ракета является многоступенчатой (т.е. появляются условия отброса ступени по исчерпании запаса топлива, задающиеся ограничениями в неизвестные моменты времени внутри отрезка интегрирования).

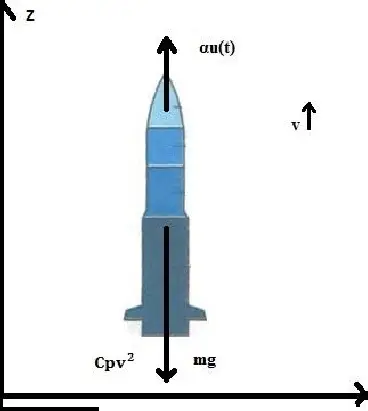
В нашей задаче мы использовали упрощенную задачу Годдарда – без учета гравитационного поля и для одноступенчатой ракеты.

**Постановка задачи**

Ракету приводят в движение, активировав запал, от чего в камере сгорания вспыхивает топливо и начинает, сгорая, переходить в газ, который, выходя под давлением из сопла, толкает ракету. По изначальному условию запас топлива истощается через 20 секунд и после этого ракета находится в свободном полёте, продолжая по инерции набирать высоту. Вскоре, ракета достигает наивысшей точки своей траектории и начинает падать. Полёт ракеты рассматривается на интервале от 0 до 100 секунд.

Задача заключается в том, чтобы найти сочетание параметров, при котором ракета наберёт максимальную высоту.

**Качественная модель**



На данном рисунке показаны все силы, действующие на ракету. По 2 закону Ньютона сумма всех сил равняется F = ma. Используя наш пример, запишем уравнение:

Разделим уравнение на m:

Так как ускорение является первой производной от скорости, получим:

Скорость уменьшения массы, где u(t) – скорость сгорания топлива:

Скорость движения ракеты вычисляется:

Итак, мы получили систему уравнений:

**Параметры задачи:**

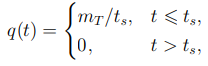
1. Так как большую часть массы ракеты составляет топливо, которое с момента запуска двигателя выгорает, то она рассчитывается по формуле:

, где **–** расход топлива

1. Набор высоты ракетой напрямую зависит от скорости:
2. Скорость зависит от множества параметров, где **𝘨** – ускорение силы тяготения, **α** – постоянная, характеризующая зависимость силы тяги от расхода топлива, **Сρυ²** – лобовое сопротивление, **C = const** – характеризует взаимодействие ракеты с обтекающим потоком,

**ρ = exp(−γz²)** – плотность воздуха, **γ = const** – характеризует убывание плотности атмосферы с увеличением высоты:

1. Зависимость расхода топлива от времени, где – масса топлива в начальный момент времени, – время полного выгорания топлива:

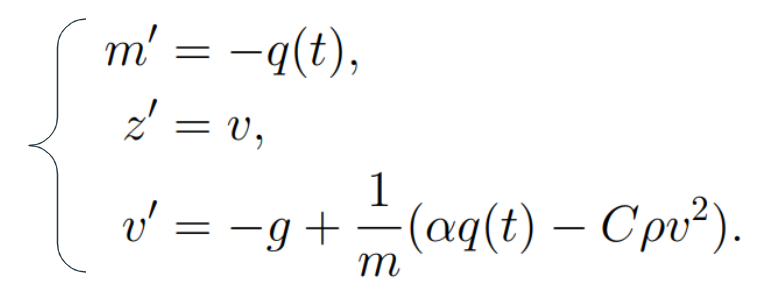


**Начальные условия:**

1. В момент запуска ракеты параметры имеют условия:
2. Константы:
3. Время полёта ракеты:

**Математическая модель**

Нам были даны параметры m, z, v, из которых нужно составить систему дифференциальных уравнений.



**Программная реализация системы зависимости параметров вертикального подъёма ракеты от времени:**

def Phi(t, y):

m, z, u = y

w1 = -q(t, t\_s)

w2 = u

w3 = -g + 1/m \* (d\*q(t, t\_s) - C\*p(z) \* u \*\* 2)

return [w1, w2, w3]

**Программная реализация зависимости расхода топлива от времени:**

def q(t, ts):

if t <= ts:

qu = m\_t/ts

else:

qu = 0

return qu

**Метод Рунге-Кутта**

Для решения системы ОДУ первого порядка требуется решить задачу Коши, которую мы решаем численным методом Рунге-Кутта 2-го порядка с начальными условиями .

Данный метод в нашей программе реализован с помощью внешней библиотеки Python scipy.integrate.

**Программная реализация:**

nSol = solve\_ivp(Phi, [t0, tMax], x0, method=nMethod, t\_eval=tEval)

**Оптимизация вертикального подъёма ракеты**

Основная задача нашей работы – нахождение оптимального времени сгорания топлива для максимизации подъёма ракеты за интервал в 100 секунд с начала запуска.

Оптимальное время сгорания топлива будет искаться на отрезке от 3 до 30 секунд.

Для нахождения оптимального значения будет использоваться минимизации функции одной переменной, используя методы решения золотого сечения, парабол и комбинированный метод Брента. **Метод золотого сечения** – метод поиска экстремума действительной функции одной переменной на заданном отрезке. В основе метода лежит принцип деления отрезка в пропорциях золотого сечения. **Метод парабол** – метод, предлагающий аппроксимировать функцию *f(x)* с помощью квадратичной функции. **Комбинированный метод Брента** – комбинирует в себе два вышеперечисленных метода.

**Полученные результаты массы m(t) высоты z(t) и скорости u(t)**

Значения параметров в каждый момент времени на интервале T = [0,100] при t\_s = 20:

t m(t) z(t) u(t)

0.000 1.0000e+00 0.0000e+00 0.0000e+00

1.010 9.5960e-01 3.6250e-02 7.2295e-02

2.020 9.1919e-01 1.4705e-01 1.4755e-01

3.030 8.7879e-01 3.3521e-01 2.2537e-01

4.040 8.3838e-01 6.0309e-01 3.0524e-01

5.051 7.9798e-01 9.5238e-01 3.8654e-01

6.061 7.5758e-01 1.3843e+00 4.6858e-01

7.071 7.1717e-01 1.8991e+00 5.5072e-01

8.081 6.7677e-01 2.4965e+00 6.3236e-01

9.091 6.3636e-01 3.1748e+00 7.1299e-01

10.101 5.9596e-01 3.9339e+00 7.9311e-01

11.111 5.5556e-01 4.7746e+00 8.7336e-01

12.121 5.1515e-01 5.6973e+00 9.5445e-01

13.131 4.7475e-01 6.7030e+00 1.0386e+00

14.141 4.3434e-01 7.7975e+00 1.1294e+00

15.152 3.9394e-01 8.9880e+00 1.2305e+00

16.162 3.5354e-01 1.0289e+01 1.3481e+00

17.172 3.1313e-01 1.1720e+01 1.4898e+00

18.182 2.7273e-01 1.3311e+01 1.6664e+00

19.192 2.3232e-01 1.5104e+01 1.8930e+00

20.202 1.9975e-01 1.7152e+01 2.1123e+00

21.212 1.9975e-01 1.9257e+01 2.0613e+00

22.222 1.9975e-01 2.1324e+01 2.0332e+00

23.232 1.9975e-01 2.3369e+01 2.0158e+00

24.242 1.9975e-01 2.5398e+01 2.0028e+00

25.253 1.9975e-01 2.7416e+01 1.9918e+00

26.263 1.9975e-01 2.9422e+01 1.9812e+00

27.273 1.9975e-01 3.1418e+01 1.9710e+00

28.283 1.9975e-01 3.3404e+01 1.9609e+00

29.293 1.9975e-01 3.5380e+01 1.9508e+00

30.303 1.9975e-01 3.7345e+01 1.9407e+00

31.313 1.9975e-01 3.9300e+01 1.9306e+00

32.323 1.9975e-01 4.1245e+01 1.9205e+00

33.333 1.9975e-01 4.3180e+01 1.9104e+00

34.343 1.9975e-01 4.5105e+01 1.9003e+00

35.354 1.9975e-01 4.7019e+01 1.8902e+00

36.364 1.9975e-01 4.8923e+01 1.8801e+00

37.374 1.9975e-01 5.0817e+01 1.8700e+00

38.384 1.9975e-01 5.2701e+01 1.8599e+00

39.394 1.9975e-01 5.4574e+01 1.8498e+00

40.404 1.9975e-01 5.6438e+01 1.8397e+00

41.414 1.9975e-01 5.8291e+01 1.8296e+00

42.424 1.9975e-01 6.0134e+01 1.8194e+00

43.434 1.9975e-01 6.1966e+01 1.8093e+00

44.444 1.9975e-01 6.3789e+01 1.7992e+00

45.455 1.9975e-01 6.5601e+01 1.7891e+00

46.465 1.9975e-01 6.7403e+01 1.7790e+00

47.475 1.9975e-01 6.9195e+01 1.7689e+00

48.485 1.9975e-01 7.0977e+01 1.7588e+00

49.495 1.9975e-01 7.2749e+01 1.7487e+00

50.505 1.9975e-01 7.4510e+01 1.7386e+00

51.515 1.9975e-01 7.6261e+01 1.7285e+00

52.525 1.9975e-01 7.8002e+01 1.7184e+00

53.535 1.9975e-01 7.9733e+01 1.7083e+00

54.545 1.9975e-01 8.1453e+01 1.6982e+00

55.556 1.9975e-01 8.3163e+01 1.6881e+00

56.566 1.9975e-01 8.4863e+01 1.6780e+00

57.576 1.9975e-01 8.6553e+01 1.6679e+00

58.586 1.9975e-01 8.8233e+01 1.6578e+00

59.596 1.9975e-01 8.9902e+01 1.6477e+00

60.606 1.9975e-01 9.1562e+01 1.6376e+00

61.616 1.9975e-01 9.3211e+01 1.6275e+00

62.626 1.9975e-01 9.4850e+01 1.6174e+00

63.636 1.9975e-01 9.6478e+01 1.6073e+00

64.646 1.9975e-01 9.8097e+01 1.5972e+00

65.657 1.9975e-01 9.9705e+01 1.5871e+00

66.667 1.9975e-01 1.0130e+02 1.5770e+00

67.677 1.9975e-01 1.0289e+02 1.5669e+00

68.687 1.9975e-01 1.0447e+02 1.5568e+00

69.697 1.9975e-01 1.0604e+02 1.5467e+00

70.707 1.9975e-01 1.0759e+02 1.5366e+00

71.717 1.9975e-01 1.0914e+02 1.5265e+00

72.727 1.9975e-01 1.1068e+02 1.5164e+00

73.737 1.9975e-01 1.1220e+02 1.5063e+00

74.747 1.9975e-01 1.1372e+02 1.4962e+00

75.758 1.9975e-01 1.1523e+02 1.4861e+00

76.768 1.9975e-01 1.1672e+02 1.4760e+00

77.778 1.9975e-01 1.1821e+02 1.4659e+00

78.788 1.9975e-01 1.1968e+02 1.4558e+00

79.798 1.9975e-01 1.2115e+02 1.4457e+00

80.808 1.9975e-01 1.2260e+02 1.4356e+00

81.818 1.9975e-01 1.2405e+02 1.4255e+00

82.828 1.9975e-01 1.2548e+02 1.4154e+00

83.838 1.9975e-01 1.2691e+02 1.4053e+00

84.848 1.9975e-01 1.2832e+02 1.3952e+00

85.859 1.9975e-01 1.2973e+02 1.3851e+00

86.869 1.9975e-01 1.3112e+02 1.3750e+00

87.879 1.9975e-01 1.3251e+02 1.3649e+00

88.889 1.9975e-01 1.3388e+02 1.3548e+00

89.899 1.9975e-01 1.3524e+02 1.3447e+00

90.909 1.9975e-01 1.3660e+02 1.3346e+00

91.919 1.9975e-01 1.3794e+02 1.3245e+00

92.929 1.9975e-01 1.3927e+02 1.3144e+00

93.939 1.9975e-01 1.4059e+02 1.3043e+00

94.949 1.9975e-01 1.4191e+02 1.2942e+00

95.960 1.9975e-01 1.4321e+02 1.2841e+00

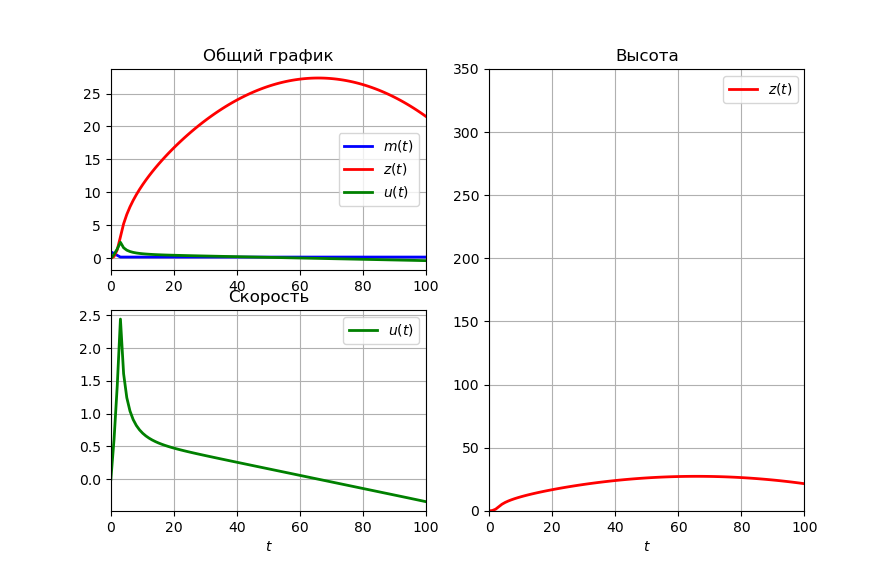
96.970 1.9975e-01 1.4450e+02 1.2740e+00

97.980 1.9975e-01 1.4578e+02 1.2639e+00

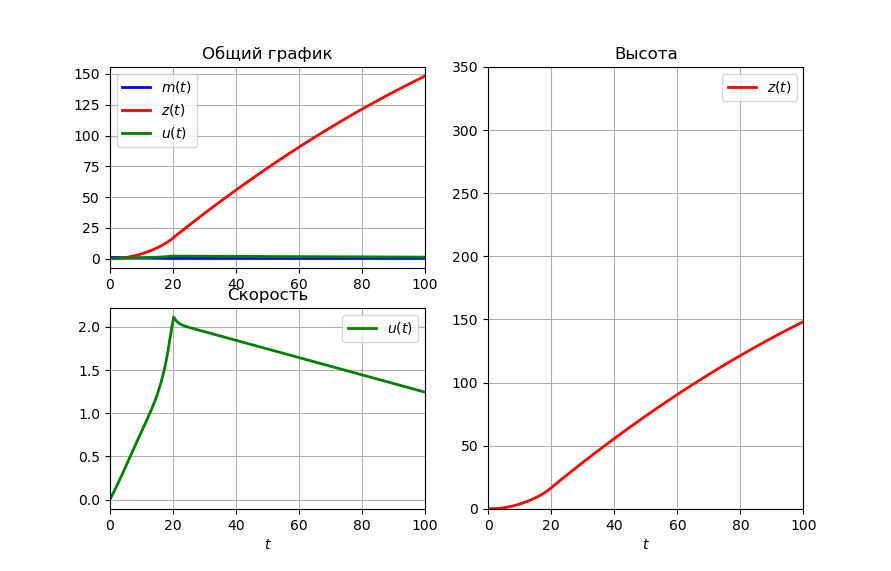
98.990 1.9975e-01 1.4705e+02 1.2538e+00

100.000 1.997e-01 1.4832e+02 1.2437e+00

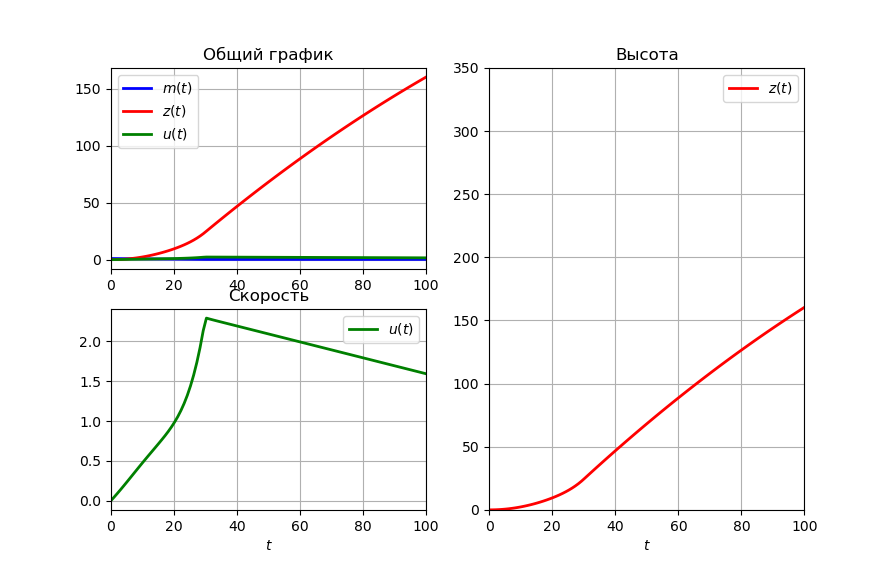
**Графики**



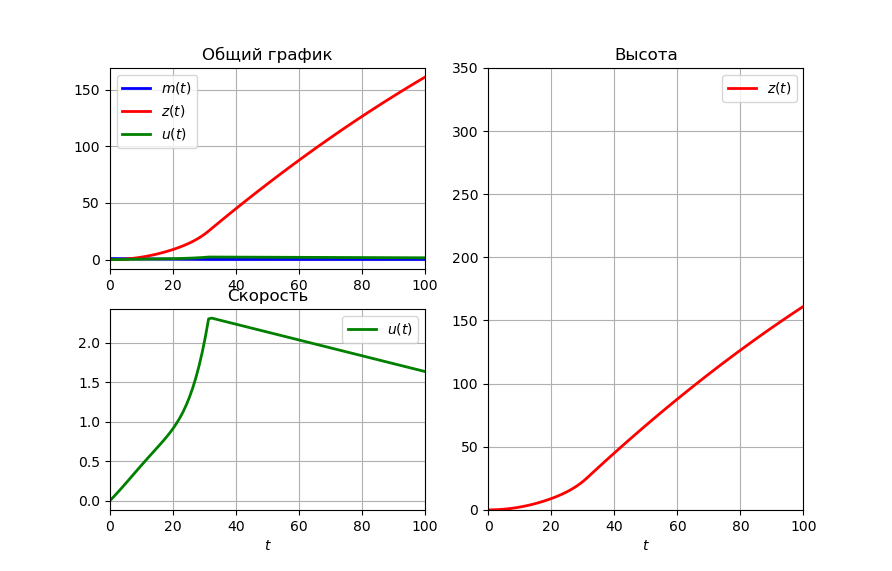
Зависимость параметров от времени при , рис. 1



Зависимость параметров от времени при , рис. 2



Зависимость параметров от времени при , рис. 3



Зависимость параметров от времени при оптимальном значении , рис. 4

**Вывод**

Решив систему ОДУ и минимизировав полученные функции, программа вывела оптимальное время сгорания топлива, равное 31-ой секунде, благодаря чему ракета поднялась на максимальную высоту.

**Листинг**

**Код:**

#!/usr/bin/python3

# -\*- coding: utf-8 -\*-

from time import time

import math

import numpy as np

from scipy.integrate import solve\_ivp, simps

from scipy.optimize import minimize, minimize\_scalar

import matplotlib.pyplot as plt

# Константы

q\_t = 0.04

g = 0.01

d = 2

C = 0.05

Y = 0.01

t\_s = 20

m\_t = 0.8

# Начальные условия

x0, t0 = [1.0, 0.0, 0.0], 0.0

# Временные границы

tStart = 0.0

tMax = 100.0

# Определение точек, в которых производятся вычисления

nEval = 100

tEval = np.linspace(t0, tMax, nEval)

# Метод решения ОДУ(Рунге Кутта)

nMethod = 'RK23'

# Зависимость расхода топлива от времени

def q(t, ts):

if t <= ts:

try:

qu = m\_t / ts

except ZeroDivisionError:

qu = m\_t

else:

qu = 0

return qu

# Константа плотности воздуха

p = lambda z: np.exp(-Y \* z \*\* 2)

# Система ОДУ (правая часть)

def Phi(t, y):

m, z, u = y

w1 = -q(t, t\_s)

w2 = u

w3 = -g + 1 / m \* (d \* q(t, t\_s) - C \* p(z) \* u \*\* 2)

return [w1, w2, w3]

# Решение сисстемы ОДУ

def DSolveProblem(printResult=True, tempStart=0.0):

# Изменяем значение глобальной переменной t\_s

global t\_s

t\_s = tempStart

# Засечем время начала расчетов

tic1 = time()

# Решаем систему ОДУ

nSol = solve\_ivp(Phi, [t0, tMax], x0, method=nMethod, t\_eval=tEval)

tSol = nSol.t

xSol = nSol.y

# Подсчитаем время выполнения расчетов

tic2 = time()

if printResult:

print('Метод - {0}\nВремя расчетов (сек.):{1}\nКоличество точек:{2}'.

format(nMethod, tic2 - tic1, len(nSol.t)))

if nSol.status == -1:

print('Интегрироваие привело к сбою.')

elif nSol.status == 0:

print('Решатель(solver) успешно достиг конца интервала.')

elif nSol.status == 1:

print('Интегрирование было завершено.')

return tSol, xSol

# Графики решения

def PlotResult(tSol, xSol):

fig = plt.figure(facecolor='white', figsize=(1.4 \* 6.4, 1.2 \* 4.8))

# Общий график

plt.subplot(2, 2, 1)

plt.plot(tSol, xSol[0, :], '-b', label=r'$m(t)$', linewidth=2)

plt.plot(tSol, xSol[1, :], '-r', label=r'$z(t)$', linewidth=2)

plt.plot(tSol, xSol[2, :], '-g', label=r'$u(t)$', linewidth=2)

plt.xlim(left=t0, right=tMax)

plt.legend(loc='best')

plt.title("Общий график")

plt.grid(True)

# График Скорости

plt.subplot(2, 2, 3)

plt.plot(tSol, xSol[2, :], '-g', label=r'$u(t)$', linewidth=2)

plt.xlim(left=t0, right=tMax)

plt.legend(loc='best')

plt.xlabel(r"$t$")

plt.title("Скорость")

plt.grid(True)

# График высоты подъема

plt.subplot(1, 2, 2)

plt.plot(tSol, xSol[1, :], '-r', label=r'$z(t)$', linewidth=2)

plt.xlim(left=t0, right=tMax)

plt.ylim(bottom=0.0, top=350.0)

plt.legend(loc='best')

plt.xlabel(r"$t$")

plt.title("Высота")

plt.grid(True)

pass

# Определим функцию, которая вычисляет выход продукта

def x3Max(var\_tStart):

tSol, xSol = DSolveProblem(printResult=False, tempStart=var\_tStart)

return -simps(xSol[2:], tSol)

# Main

if \_\_name\_\_ == '\_\_main\_\_':

tSol, xSol = DSolveProblem(tempStart=3.0)

PlotResult(tSol, xSol)

tSol, xSol = DSolveProblem(tempStart=20.0)

PlotResult(tSol, xSol)

tSol, xSol = DSolveProblem(tempStart=30.0)

PlotResult(tSol, xSol)

plt.show() # display

# Находим минимумум функции -ВыходПродукта

res = minimize\_scalar(x3Max, bounds=(t0, tMax), method='Bounded',

options={'xatol': 1e-09, 'maxiter': 500, 'disp': 3})

print('Нахождение оптимального времени сгорания топлива для максимального взлета "z"\n', res)

tSol, xSol = DSolveProblem(printResult=False, tempStart=res.x)

PlotResult(tSol, xSol)

plt.show() # display

print(x3Max(0.0))

**Вывод программы:**

Метод - RK23

Время расчетов (сек.):0.6454076766967773

Количество точек:100

Решатель(solver) успешно достиг конца интервала.

Метод - RK23

Время расчетов (сек.):0.0039882659912109375

Количество точек:100

Решатель(solver) успешно достиг конца интервала.

Метод - RK23

Время расчетов (сек.):0.004950046539306641

Количество точек:100

Решатель(solver) успешно достиг конца интервала.

Func-count x f(x) Procedure

1 38.1966 -157.182 initial

2 61.8034 -125.344 golden

3 23.6068 -155.413 golden

4 32.4767 -160.095 parabolic

5 31.7544 -160.124 parabolic

6 31.8374 -159.895 parabolic

7 28.6423 -159.908 golden

8 31.4095 -161.091 parabolic

9 30.3526 -160.877 golden

10 30.9282 -161.037 parabolic

11 31.5332 -160.742 parabolic

12 31.2257 -160.843 golden

13 31.3393 -160.535 golden

14 31.4568 -160.958 golden

15 31.3827 -161.167 golden

16 31.3661 -160.461 golden

17 31.393 -161.138 golden

18 31.3764 -161.184 golden

19 31.3725 -161.195 golden

20 31.3701 -160.45 golden

21 31.374 -161.191 golden

22 31.3715 -161.198 golden

23 31.371 -161.2 golden

24 31.3706 -160.448 golden

25 31.3712 -161.199 golden

26 31.3708 -160.448 golden

27 31.3711 -161.199 golden

28 31.3709 -161.2 golden

29 31.3709 -161.2 golden

30 31.3709 -161.2 golden

31 31.3709 -160.448 golden

32 31.3709 -161.2 golden

33 31.3709 -161.2 golden

34 31.3709 -161.2 golden

35 31.3709 -161.2 golden

36 31.3709 -161.2 golden

37 31.3709 -160.448 golden

38 31.3709 -161.2 golden

Optimization terminated successfully;

The returned value satisfies the termination criteria

(using xtol = 1e-09 )

Нахождение оптимального времени сгорания топлива для максимального взлета "z"

fun: array([-161.19983379])

message: 'Solution found.'

nfev: 38

status: 0

success: True

x: array([31.3708606])

[60.14572926]