Национальный исследовательский университет «МЭИ»

ЛАБОРАТОРНАЯ РАБОТА №4

«Определение настроечных параметров регулятора методом

автоматического подбора»

Выполнил: Максимов Р.С.

Москва 2019 г.

#### **Цель работы**

Определение показателей качества стационарной системы автоматического управления в установившихся и переходных режимах прямыми и косвенными методами.

1. **Задание**

1. Необходимо определить передаточную функцию АСР

2. Определить параметры регулятора для соответствия переходного процесса заданным критериям качества, приведенных в таблице 1.

3. Определить прямые оценки качества переходного процесса:

4. По распределению корней на комплексной плоскости замкнутой САУ определить.

5. По логарифмическим частотным характеристикам определить.

6. По интегральному методу оценить степень качества системы регулирования

1. **Исходные данные**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вариант | Время регулирования, с | Показатель колебательности | Перерегулирование,% | Тип регулятора | Метод |
| 14 | 13 | 1.20 | 25 | ПИД | Градиентный спуск |

1. **Анализ параметров ПИД-регулятора**
   1. **Прямой метод+-**

На рисунке 1 представлена структурная схема САУ. Для данной схемы на рисунке 2 представлена переходная характеристика.



Рисунок 1. Структурная схема АСР

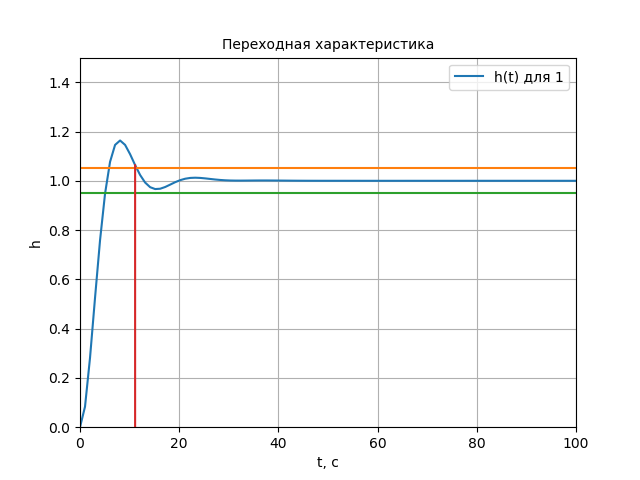
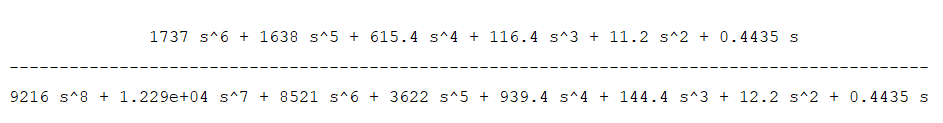
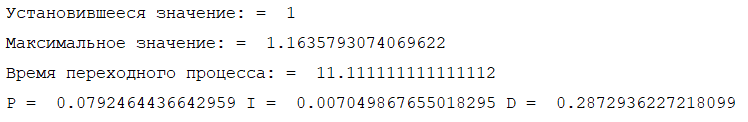


Рисунок 2. Переходная характеристика.

Передаточная функция:



Вывод программы:



* 1. **Корневой метод**

На рисунке 3 представлено распределение корней характеристического уравнения на координатной плоскости:

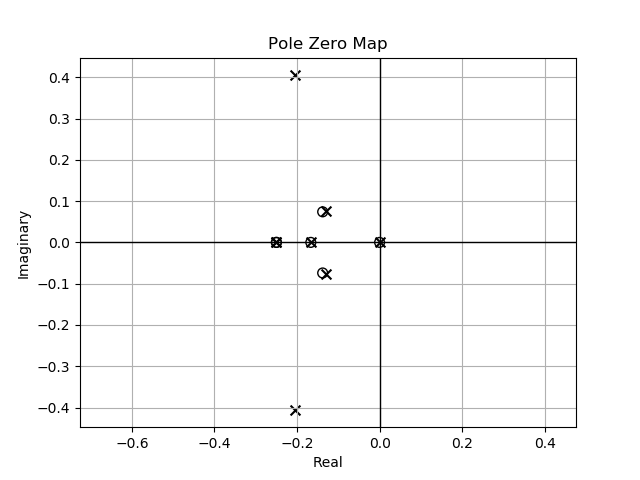
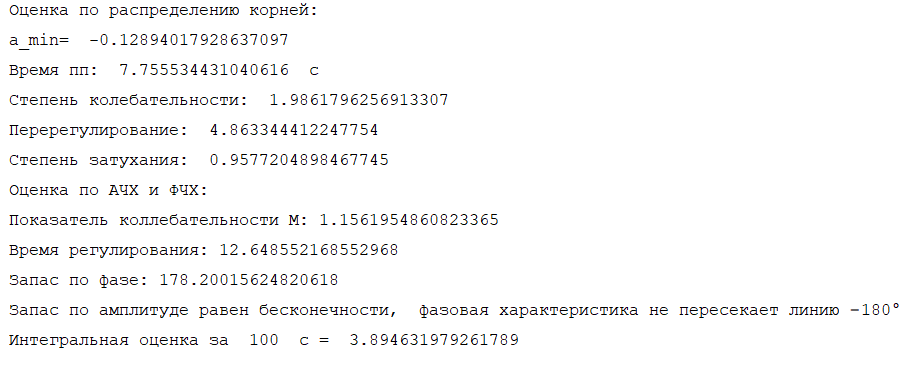


Рисунок 3- Корни характеристического уравнения.

Вывод программы:



**с. По логарифмическим частотным характеристикам**

На рисунке 4 представлены частотные характеристики для данной САУ (ЛАЧХ и ЛФЧХ).

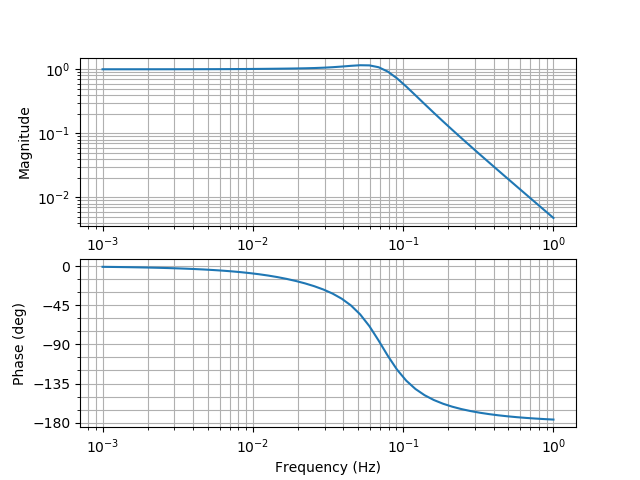
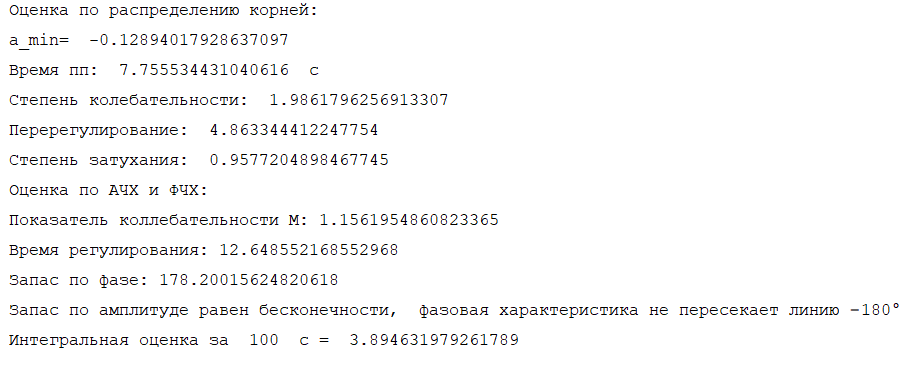
****

Рисунок 4 - ЛАЧХ и ЛФЧХ системы САУ.

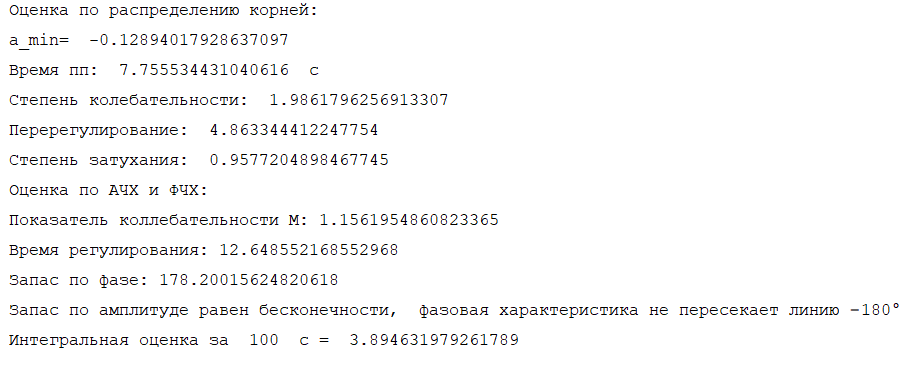
Вывод программы:



1. **Интегральный метод**

Интегральные оценки качества представляют собой интегралы по времени (в пределах от 0 до ) от некоторой функции управляемой переменной или сигнала ошибки . Интегральные оценки не связаны непосредственно с количественными оценками переходных процессов. Они чаще всего применяются в качестве функционала оптимизации по быстродействию систем автоматического управления. Простейшей интегральной оценкой является линейная интегральная оценка, которая равна площади, заключенной между прямой X(∞) и кривой X(t).

Вывод программы:



# Приложение А: код программы.

import numpy as np  
import matplotlib.pyplot as plt  
import control.matlab as con  
import control as c  
import sympy as sp  
import time  
from datetime import timedelta  
from numba import jit  
import math  
import scipy.integrate as integrate  
import time as tttt  
  
# @jit  
def nyquis(W):  
 plt.grid()  
 c.nyquist\_plot(W)  
 plt.show()  
  
  
def pzmap(W, T):  
 # f = con.pzmap(W).  
 g = np.array(W.pole()) # возвращает список всех полюсов - корней знаменателя  
 # plt.title("Расположение полюсов на комплексной плоскости")  
 # plt.grid(True)  
 # print("Корни = ", g)  
 plt.close()  
 D = True  
 j = 0  
 G = False  
 kol = 0  
 a = np.array(np.zeros((1000, 2)))  
 # print(g)  
 while D and j < len(g):  
 r = g[j].real  
 # print(r)  
 if float(r) <= 0.000000:  
 # print("Левый корень = " + str(r))  
 D = True  
 if float(r) == 0.0:  
 G = True  
 else:  
 a[kol, 0] = np.abs(r)  
 a[kol, 1] = np.abs(g[j].imag)  
 kol += 1  
 else:  
 D = False  
 # print("Первый правый корень = " + str(r))  
 j = j + 1  
 # if D and G:  
 # print("Система на границе устойчивости, есть нулевые корни")  
 # elif D:  
 # print("Система устойчива, все корни левые")  
 # else:  
 # print("Cистема не устойчива, есть хотя бы один правый корень")  
 a = a[:kol, :]  
 print(a)  
 # plt.title("Расположение полюсов на комплексной плоскости")  
 # plt.grid(True)  
 # plt.show()  
 if T:  
 return D  
 else:  
 return a  
  
  
# @jit  
def mihalych(W1):  
 P2 = list(W1.den[0])  
 a = P2[0] # коэфициенты  
 plt.figure(1)  
 w = sp.symbols(**"w"**, real=True) # Годограф Михайлова  
 z = 0  
 for i in range(0, len(a), 1):  
 z = z + a[i] \* (1j \* w) \*\* ((len(a) - 1) - i)  
 z = sp.factor(z)  
 zR = sp.re(z)  
 zIm = sp.im(z)  
 x = [zR.subs({w: q}) for q in np.arange(0, 100, 0.01)]  
 y = [zIm.subs({w: q}) for q in np.arange(0, 100, 0.01)]  
 plt.title(**"Годограф Михайлова"**)  
 plt.axis([-100, 30, -150, 15])  
 plt.plot(x, y)  
 plt.grid()  
 plt.show()  
  
  
**""" Функция, необходимая для Гурьвица """**#  
# @jit  
def matrica\_from\_spisok(z, k): # k - kolichestvo strok=stolbcov  
 matr = **''** z = z  
 k = k  
 h = 1  
 for i in range(0, len(z), 1):  
 if ((h \* k) - i) == 0 and i != 0 and i != len(z) - 1: # последний?  
 matr = matr + **';'** + str(z[i]) + **' '** h = h + 1  
 else:  
 if i == len(z) - 1: # проверка на самый посл элемент  
 matr = matr + str(z[i])  
 else:  
 matr = matr + str(z[i]) + **' '** Mat = np.matrix(matr)  
 return Mat  
  
  
**""" Функция, необходимая для Гурьвица """**# @jit  
def matrica\_spisok(z1, z2, k):  
 """ к - количество столбцов и строк в матрице, ранг матрицы  
 z1 - список четных коэф  
 z2 - список нечетных коэф  
 """  
 z = []  
 h1 = 0  
 h2 = 0  
 for g in range(0, k, 1):  
 index\_nach\_nech = h1  
 index\_nach\_ch = h2  
 index\_konec\_nech = k - len(z1) - h1  
 index\_konec\_ch = k - len(z2) - h2  
 if g % 2 == 0: # no четный  
 for i in range(0, index\_nach\_nech, 1):  
 z.append(**"0"**)  
 for i in range(0, len(z1), 1):  
 z.append(z1[i])  
 for i in range(0, index\_konec\_nech, 1):  
 z.append(**"0"**)  
 h1 = h1 + 1  
 else:  
 for i in range(0, index\_nach\_ch, 1):  
 z.append(**"0"**)  
 for i in range(0, len(z2), 1):  
 z.append(z2[i])  
 for i in range(0, index\_konec\_ch, 1):  
 z.append(**"0"**)  
 h2 = h2 + 1  
 return z  
  
  
**""" Критерий Гурвица. Все корни полинома Δ(s) имеют отрицательные вещественные части то-  
гда и только тогда, когда все n главных миноров матрицы А (определителей Гурвица) поло-  
жительны. """**def gurych(W1):  
 T = True  
 P2 = (W1.den[0])  
 a = P2[0] # коэфициенты  
 z1 = []  
 z2 = []  
 for i in range(0, len(a), 1):  
 if (i % 2 != 0):  
 z1.append(str(a[i]))  
 else:  
 z2.append(str(a[i]))  
  
 # print(z1) #нечетные коэф  
 # print(z2) #четные коэф  
 kol\_str\_stol = len(a) - 1 # количество строк и столбцов  
 z = matrica\_spisok(z1, z2, kol\_str\_stol) # создаем матрицу n порядка  
 Mat\_gl = matrica\_from\_spisok(z, kol\_str\_stol)  
 Del\_mat = np.linalg.det(Mat\_gl)  
  
 **""" необходимо рассмотреть определители главных миноров матрицы"""** Del\_minors = []  
 for i in range(1, kol\_str\_stol, 1):  
 Mat\_\_minor\_gl = Mat\_gl[:i, :i]  
 Del\_mat\_minor = np.linalg.det(Mat\_\_minor\_gl)  
 Del\_minors.append(Del\_mat\_minor)  
  
 **"""Проверка определителей главных миноров"""** for i in Del\_minors:  
 if i < 0:  
 T = False  
 # if (T):  
 # print("Определитель главной матрицы = " + str(Del\_mat))  
 # print("Определители миноров = " + " ".join(str(value) for value in Del\_minors), sep=', ')  
 # else:  
 # print("Система неустойчива")  
 return T  
  
  
def pereh(W, t):  
 plt.figure(1) # Вывод графиков в отдельном окне  
 y1, t = con.step(W, t)  
 lines = [y1]  
 lines[0] = plt.plot(t, y1, **"r"**)  
 plt.legend(lines[0], [**'h(t) для 1'**], loc=**'best'**, fontsize=10)  
 plt.title(**'Переходная характеристика'**, fontsize=10)  
 plt.ylabel(**'h'**)  
 plt.xlabel(**'t, c'**)  
 plt.grid()  
 plt.show()  
  
  
def PID(Kp,Ki,Kd):  
 p = con.tf(Kp, 1) + con.tf([0, Ki], [1, 0]) + con.tf([Kd, 0], 1)  
 return p  
  
  
def ust(spisok\_y1):  
 state = float(spisok\_y1[len(spisok\_y1) - 1])  
 if (state > 0.95 and state < 1.05):  
 Valueofstate = 1  
 else:  
 Valueofstate = 0  
  
 return Valueofstate  
  
  
def transition\_time\_of\_proсess\_andGraph(y1, state, t1, Graph):  
 c = 0  
 u = 0  
 T = True  
 y = list(y1)  
 y.reverse()  
 t = list(t1)  
 t.reverse()  
 line1 = 1.05 \* state  
 line2 = 0.95 \* state  
 while (T and c < len(y)):  
 if (float(y[c]) < line1) and (float(y[c]) > line2):  
 c += 1  
 else:  
 T = False  
 transition\_time = t[c]  
 if (not T) and Graph:  
 # y.reverse()  
 line11 = [line1 for i in range(len(y1))]  
 line22 = [line2 for i in range(len(y1))]  
 line3 = np.arange(0, y[c], 0.001)  
 c = np.ones((1, len(line3))) \* t[c]  
 spisok\_time = c[0]  
 lines = [y, line11, line22, line3]  
 lines[0], lines[1], lines[2], lines[3] = plt.plot(t, y, t, line11, t, line22, spisok\_time, line3)  
 plt.legend(lines, [**'h(t) для 1'**], loc=**'best'**, fontsize=10)  
 plt.title(**'Переходная характеристика'**, fontsize=10)  
 plt.ylabel(**'h'**)  
 plt.xlabel(**'t, c'**)  
 plt.axis([0, 100, 0, 1.5])  
 plt.grid()  
 plt.show()  
 return transition\_time  
  
  
def search\_order\_numbers(grad):  
 bool = True  
 first = 10  
 number\_order = 0  
 grag\_int = int(grad)  
 while bool:  
 relation = grag\_int / first  
 if (np.abs(relation) > 1):  
 number\_order += 1  
 first = first \* 10  
 else:  
 bool = False  
  
 return number\_order + 1  
  
  
def Pole(W):  
 con.pzmap(W)  
 plt.plot()  
 plt.grid(True)  
 plt.show()  
 print(**'Оценка по распределению корней:'**)  
 Pol = con.pole(W)  
 # print(Pol)  
 P = []  
 **""" Показатель колебательности характеризует склонность системы к  
 колебаниям: чем выше М, тем менее качественна система при прочих  
 равных условиях. Считается допустимым, если 1,1 < М < 1,5. """** degreeovershoot\_M = []  
 for i in Pol:  
 k = complex(i)  
 if k.real != 0:  
 P.append(k.real)  
 m = k.imag / k.real  
 degreeovershoot\_M.append(m)  
 a\_min = max(P)  
 t\_reg = abs(1 / a\_min)  
 overshooting = math.exp(math.pi / max(degreeovershoot\_M))  
 psi = 1 - math.exp((-2) \* math.pi / max(degreeovershoot\_M))  
 print(**"a\_min= "**, a\_min)  
 print(**"Время пп: "**, t\_reg, **" c"**)  
 print(**"Степень колебательности: "**, max(degreeovershoot\_M))  
 print(**"Перерегулирование: "**, overshooting)  
 print(**"Степень затухания: "**, psi)  
  
  
def Freq(W):  
 print(**"Оценка по АЧХ и ФЧХ:"**)  
 # t = np.linspace(0, stop=100, num=1000)  
 mag, phase, omega = con.bode(W, dB=False)  
 plt.plot()  
 # plt.close()  
 plt.show()  
 mag\_max = max(mag)  
 M = (mag\_max / mag[0])  
 print(**"Показатель коллебательности М: "** + str(M))  
 t = True  
 n = list(mag).index(mag\_max)  
 while n < len(mag) and t:  
 if ((mag[n] > mag[0] - 0.01) and (mag[n] < mag[0] + 0.01)) or mag[n] < mag[0]:  
 wc = omega[n]  
 phase\_res = phase[n]  
 print(**"Время регулирования: "** + str(2 \* math.pi / wc))  
 **""" В хорошо демпфированных системах запас устойчивости по амплитуде колеблется   
 в пределах от 6 до 20 дБ, а запас по фазе от 30 до 60"""** print(**"Запас по фазе: "** + str(180 - abs(phase\_res)))  
 t = False  
 n = n + 1  
  
 t\_magn = True  
 n = 0  
 while n < len(phase) and t\_magn:  
 if -179.7 > phase[n] > -180.7:  
 index\_phase\_180 = list(phase).index(phase[n])  
 print(**"Запас по амплитуде: "** + str(round(mag[0] - mag[index\_phase\_180], 3)))  
 t\_magn = False  
 n = n + 1  
 **"""запас по амплитуде может быть равен бесконечности, если фазовая характеристика не   
 пересекает линию −180°"""** if t\_magn:  
 print(**"Запас по амплитуде равен бесконечности, фазовая характеристика не пересекает линию −180°"**)  
  
  
def intergalnaya\_Otsenka(W):  
 a = 0  
 b = 100  
 n = 1000  
 h = (b - a) / n  
 t = np.linspace(a, b, num=n)  
 y, x = con.step(W, t) # х-время ПП  
 func = 0  
 x0 = x[0]  
 y0 = y[0]  
 # нахождение площади методом трапеций  
 for i in range(0, len(x), 1):  
 xi = x[i]  
 y1 = y[i]  
 func += abs(1 \* (xi - x0) - 0.5 \* (y1 + y0) \* (xi - x0))  
 x0 = xi  
 y0 = y1  
 # print(func)  
 print(**"Интегральная оценка за "**, b, **" c = "**, func)  
  
  
def for\_iter(coef\_regul, main\_condition, middle\_condition, delta\_overshoot, delta\_processtime, name):  
 alfa = []  
 for i in range(0, 50):  
 ran = np.random.random()  
 if i < 2:  
 alfa.append(ran \* 0.00001)  
 elif i < 4:  
 alfa.append(ran \* 0.001)  
 elif i < 5:  
 alfa.append(ran \* 0.01)  
 elif i < 10:  
 alfa.append(ran \* 1)  
 # print(alfa)  
 grad\_overshoot = delta\_overshoot + 0.0001  
 grad\_processtime = delta\_processtime + 0.0001  
 print(**"Градиент для максимума="**, grad\_overshoot, **" для "** + name)  
 print(**"Градиент для времени="**, grad\_processtime, **" для "** + name)  
 number\_of\_grad\_overshoot = search\_order\_numbers(grad\_overshoot)  
 number\_of\_grad\_processtime = search\_order\_numbers(grad\_processtime)  
 alfa\_spisok = []  
 # print("HI " + name)  
 if main\_condition and (not middle\_condition) and (name == **"P"** or name == **"D"**):  
 if number\_of\_grad\_overshoot > len(alfa):  
 if (grad\_overshoot>0):  
 alfa\_forovershot = alfa[len(alfa) - 1] \* (grad\_overshoot / (np.abs(grad\_overshoot)))\*(np.abs(grad\_overshoot))  
 else:  
 alfa\_forovershot = - alfa[len(alfa) - 1] \* (grad\_overshoot / (np.abs(grad\_overshoot))) \* (  
 np.abs(grad\_overshoot))  
 else:  
 if (grad\_overshoot>0):  
 alfa\_forovershot = alfa[number\_of\_grad\_overshoot] \* grad\_overshoot / (np.abs(grad\_overshoot))\*(np.abs(grad\_overshoot))  
 else:  
 alfa\_forovershot = - alfa[number\_of\_grad\_overshoot] \* grad\_overshoot / (np.abs(grad\_overshoot)) \* (  
 np.abs(grad\_overshoot))  
 the\_best\_alfa = alfa\_forovershot  
 # print("Best11111 = " + str(the\_best\_alfa) + " " + name)  
  
 elif not main\_condition and (middle\_condition) and (name == **"P"** or name == **"D"**):  
 if number\_of\_grad\_processtime > len(alfa):  
 # if (grad\_overshoot > 0):  
 # alfa\_forprocesstime = alfa[len(alfa) - 1] \* (grad\_processtime / (np.abs(grad\_processtime)))\*(np.abs(grad\_processtime))  
 # else:  
 alfa\_forprocesstime = alfa[len(alfa) - 1] \* (grad\_processtime / (np.abs(grad\_processtime))) \* (  
 np.abs(grad\_processtime))  
 else:  
 # if grad\_processtime > 0:  
 # alfa\_forprocesstime = alfa[number\_of\_grad\_processtime] \* grad\_processtime / (np.abs(grad\_processtime))\*(np.abs(grad\_processtime))  
 # else:  
 alfa\_forprocesstime = alfa[number\_of\_grad\_processtime] \* grad\_processtime / (  
 np.abs(grad\_processtime)) \* (np.abs(grad\_processtime))  
 the\_best\_alfa = alfa\_forprocesstime  
 # print("number\_of\_grad\_processtime " + str(number\_of\_grad\_processtime) + " " + name)  
 # print("grad\_processtime " + str(grad\_processtime) + " " + name)  
 # print("Best22222 = " + str(the\_best\_alfa) + " " + name)  
 elif (not main\_condition and (middle\_condition)) or (name == **"I"**):  
 the\_best\_alfa = 0.0001  
 else:  
 the\_best\_alfa = alfa[number\_of\_grad\_processtime] \* grad\_processtime / (np.abs(grad\_processtime))\*(np.abs(grad\_processtime))  
 # print("grad\_processtime " + str(grad\_processtime) + " " + name)  
 # print("Best3333333 number\_of\_grad\_processtime= " + str(number\_of\_grad\_processtime) + " " + name)  
 # print("Best = " + str(the\_best\_alfa) + " " + name)  
 return the\_best\_alfa  
  
  
t = np.linspace(0, stop=100, num=100)  
treg = 13 # Время регулирования  
overshoot = 1.25  
  
G0 = con.tf(1, 1) # обратная связь  
G1 = con.tf(1, [4, 1]) # generator  
G2 = con.tf(3, [6, 1]) # turbine  
G3 = con.tf(21, [4, 1]) # device  
  
J = True  
p\_for\_iter = np.zeros((1, 2))  
i\_for\_iter = np.zeros((1, 2))  
d\_for\_iter = np.zeros((1, 2))  
M\_for\_iter = 0  
T\_for\_iter = 0  
M\_for = np.zeros((1, 2))  
T\_for = np.zeros((1, 2))  
p = 0.06  
i = 0.006  
d = 0.6  
# FlagP = True  
# FlagI = False  
# FlagD = False  
iter1 = 1  
start\_t= tttt.time()  
  
while J:  
 overshoot\_condition = False  
 processtime\_condition = False  
 value\_condition = False  
 p\_for\_iter[0, 1] = p  
 i\_for\_iter[0, 1] = i  
 d\_for\_iter[0, 1] = d  
 delta\_p = p\_for\_iter[0, 1] - p\_for\_iter[0, 0] + 0.00000000000001  
 delta\_i = i\_for\_iter[0, 1] - p\_for\_iter[0, 0] + 0.00000000000001  
 delta\_d = d\_for\_iter[0, 1] - p\_for\_iter[0, 0] + 0.00000000000001  
 p\_for\_iter[0, 0] = p\_for\_iter[0, 1]  
 i\_for\_iter[0, 0] = i\_for\_iter[0, 1]  
 d\_for\_iter[0, 0] = d\_for\_iter[0, 1]  
 # p\_ofpid = P(p)  
 # i\_ofpid = I(i)  
 # d\_ofpid = D(d)  
 Greg = PID(p,i,d)  
 G5 = G1 \* G2 \* G3 \* Greg  
 G\_zam = G5 / (1 + G5)  
 y1, t = con.step(G\_zam,t)  
 Value\_state = ust(y1)  
 maximom\_of\_func = max(y1)  
 time = transition\_time\_of\_proсess\_andGraph(y1, Value\_state, t, False)  
  
 M\_for\_iter = maximom\_of\_func  
 T\_for\_iter = time  
 deltaM = overshoot - M\_for\_iter  
 deltaT = treg - T\_for\_iter  
  
 if Value\_state == 1:  
 # Roots\_of\_the\_equation = pzmap(G\_zam, False)  
 # Alfa = np.array(Roots\_of\_the\_equation[:, 0])  
 # Alfa\_min = min(Alfa)  
 if maximom\_of\_func < overshoot:  
 overshoot\_condition = True  
 if time < treg:  
 processtime\_condition = True  
 if Value\_state < 1.1 and Value\_state > 0.95:  
 value\_condition = True  
  
 if (overshoot\_condition and processtime\_condition and value\_condition):  
 end\_t = tttt.time()  
 transition\_time\_of\_proсess\_andGraph(y1, Value\_state, t, True)  
 print(**'Время подбора коэф = '**, end\_t - start\_t)  
 J = False  
  
 # iter1 += 1  
 # if iter1 % 3 == 1:  
 k1 = for\_iter(delta\_p, overshoot\_condition, processtime\_condition, deltaM, deltaT, **" P"**)  
 if p + k1 > 0:  
 p = p + k1  
 else:  
 p = p+abs(k1)  
  
 # if iter1 % 3 == 2:  
 k2 = for\_iter(delta\_i, overshoot\_condition, processtime\_condition, deltaM, deltaT, **" I"**)  
 if i + k2 > 0:  
 i = i + k2  
 else:  
 i = i+abs(k2)  
  
 # if iter1 % 3 == 0:  
 k3 = for\_iter(delta\_d, processtime\_condition, processtime\_condition, deltaM, deltaT, **" D"**)  
 if d + k3 > 0:  
 d = d + k3  
 else:  
 d = d + abs(k3)  
 print(**"Установившееся значение: = "**, Value\_state)  
 print(**"Максимальное значение: = "**, maximom\_of\_func)  
 print(**"Время переходного процесса: = "**, time)  
 print(**"P = "**, p, **"I = "**, i, **"D = "**, d)  
  
print(G\_zam)  
Pole(G\_zam)  
Freq(G\_zam)  
intergalnaya\_Otsenka(G\_zam)