* 1. Код основной расчетной программы

data = read("C:\filepath", -1, 2) // Функция read считывает заданный файл и на его основании формирует числовую матрицу указанного двумя последующими числами размера (первое число — количество строк, второе — столбцов). Число -1 ставится в том случае, если точное количество структурных единиц (столбцов или строк) неизвестно, однако писать -1 одновременно вместо двух аргументов нельзя. Результат, возвращаемый этой функцией, присваивается матрице data

angle = data(:, 2)\*%pi/180 // присваиваем значения второго столбца

time = data(:,1) // -//- первого

aim = [time, angle] //создаем матрицу

aim = aim' // транспонируем

deff('e=func(k,z)', 'e=z(2) - k(1)\*(z(1) - k(2) \* (1 - exp(-z(1)/k(2))))') //записываем уравнение

att = [15; 0.06] // Задайте одностолбцовую (нельзя одностроковую) матрицу att из двух элементов, где надо разместить те значения параметров ωnls и Tm, которые, как вы думаете, получатся. Это помогает лишь ускорить процесс, поэтому содержание данной матрицы может быть любым

[koeffs, errs] = datafit(func, aim, att) // Запустите процесс аппроксимации командой [koeffs,errs] = datafit(func,aim,att). В результате ее работы будет создана матрица koeffs, в которую будут сохранены найденные параметры аппроксимирующей функции, и переменная errs, представляющая из себя сумму квадратов разностей экспериментального значения функции (angle) и найденного из уравнения аппроксимирующей функции для каждого из значений матрицы time. Процесс аппроксимации может занять некоторое время.

Wnls = koeffs(1)

Tm = koeffs(2) // Введите две новые переменные Wnls и Tm, в которые следует сохранить найденные значения параметров ωnls и Tm аппроксимирующей функции

J = 0.0023

Mst = (J\*Wnls)/Tm

model = Wnls \* (time - Tm \* (1 - exp(-time/Tm))) // Для каждого из элементов матрицы time по уравнению аппроксимирующей функции определите соответствующие ее значения и заполните ими новую матрицу model посредством команды

loadXcosLibs() //импортируем библиотеку

scs\_m = xcosDiagramToScilab("C:\filepath") // загружаем диаграмму Xcos в переменную

xcos\_simulate(scs\_m, 4) // запускаем симуляцию этой схемы, 4 – перекомпилируем каждый раз заново, не используя старые значения

plot2d(time, angle, 2)

plot2d(time, model, 3)

plot2d(A.time, A.values, 5)

plot2d(Mst, Wnls, 6)

legend('model','approximation','experiment',2)

2.4 Код программы для EV3

#!/usr/bin/env python3 // Данная строчка указывает на адрес интерпретатора питона и позволяет запускать программы из терминала со следующим синтаксисом: ./programm. Стоит отметить, что данная строчка совсем не является обязательной и без нее нужно будет запускать программы вот так: python3 programm.py

from ev3dev.ev3 import \* // Символ \* в данном случае означает, что мы импортируем все элементы находящиеся в библиотеке EV3.

import time // подключает модуль time

mA = LargeMotor('outA') //создается объект соотв мотору EV3, подкл в порт А

logs = open('logs', 'w') // создается файл в который будет записываться дата. На файл тут указывает второй флаг ‘W’ (второй арг)

logs.write('start\n') // \n - обозначает конец строки

for sp in range(-100, 101, 20):

logs.write(str(sp) + '\n')

if sp == 0:

continue

mA.position = deltaTime = 0

start\_time = time.time()

fp = open('position' + str(sp) + '.txt', 'w')

fs = open('speed' + str(sp) + '.txt', 'w')

try:

while True:

deltaTime = time.time() - start\_time

mA.run\_direct(duty\_cycle\_sp = sp) // метод запускает мотор с процентом напряжения указанным в duty\_cucle\_sp

fp.write(str(deltaTime) + ' ' + str(mA.position) + '\n')

fs.write(str(deltaTime) + ' ' + str(mA.speed) + '\n')

if deltaTime > 2:

mA.run\_direct(duty\_cycle\_sp = 0)

break

finally:

mA.stop(stop\_action = 'brake')

fp.close()

fs.close()

time.sleep(2)