МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра АПУ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Теория автоматического управления»
Тема: Идентификация нелинейных оъектов управления

Студент гр. 7392	____________________	И.В. Батура
Студент гр. 7392	____________________	Е.О. Амельченко
Студентка гр. 7392	____________________	А.Г. Цуркан
Преподаватель	____________________	А.М. Синица

Санкт-Петербург
2020

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Изучение инструментальных средств идентификации объектов управления.

2 ЗАДАЧА НА ЛАБОРАТОРНУЮ РАБОТУ

Задачи на лабораторную работу:

- построение линейной компьютерной модели, идентификация модели методом подстраиваемых параметров;
- построение компьютерной модели объекта управления с учетом нелинейности объекта, идентификация модели методом подстраиваемых параметров;
- построение нейросетевой модели объекта управления.

Для выполнения лабораторной работы нам был дан номер варианта 1, в последующем это будет использоваться в программном коде.

3 ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Первоочередной задачей при разработке системы управления является получение модели объекта управления. Существует два способа построения модели объекта: аналитический и экспериментальный. В случае аналитического способа имеются априорные знания о законах природы, которым подчиняются процессы, что позволяет описать объект без проведения экспериментов. В случае «серого» ящика о внутренней структуре объекта известна некоторая информация, но ее недостаточно для построения модели. Тогда применяют экспериментальный способ, который заключается в подаче тестовых сигналов на вход объекта, анализе реакций и обработке данных. На практике комбинируют аналитический и экспериментальный способы.

Способы идентификации далее рассмотрены без объяснения методов и алгоритмов обработки данных, которые детально изучаются в рамках дисциплины «Идентификация объектов управления». Существенным упрощением работы является отсутствие случайных возмущений. Это исключает необходимость в многократных экспериментах, хотя не вполне соответствует понятию и термину «идентификация».

Рассматриваются компьютерные имитаторы объектов с одним входом и одним выходом (типа SISO).

4 ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Учитем данный нам вариант 1, отобразим это в коде:

```
variant = 1
```

4.1 Компьютерная имитация объекта

В данном разделе мы опишем кмпобютерную модель на языке программирования Python.

Здесь описывается объект, который в рамках работы необходимо идентифицировать. Объект описывается классом, для получения функции системы ДУ необходимо вызвать фунцию getODE(), использовать полученную функцию стоит только для оценки параметров "идеальной"модели.

$$d\overline{x} = F(\overline{x})$$

```
class real_object:
def __init__(self, variant):
self._var = variant
self._ctrl_fcn = real_object._default_control
self.lin_par_1 = variant % 10 * 0.2
self.lin_par_2 = ((32 - variant) % 9 + 0.1) * 2.5
self.nonlin_par_1 = variant % 15 * 0.35
self.nonlin_par_2 = variant % 12 * 0.45
self.nonlin_fcns = [self.deadZone, self.saturation, self.relay
self.nonlin_names = ['deadZone', 'saturation', 'relay']
self.nonlin_type = variant % 3
self._params = [self.lin_par_1, self.lin_par_2, self.nonlin_par
print(self.lin_par_1, self.lin_par_2, self.nonlin_par_1, self.

def deadZone(self, x, p1, p2):
if np.abs(x) < p1:
x = 0
elif x > 0:
x = x - p1
elif x < 0:
x = x + p2
return x

def saturation(self, x, p1, p2):
if x > p1:
x = p1
elif x < -p2:
```

```
x = -p2
return x

def relay(self, x, p1, p2):
if x > 0:
return p1
else:
return -p2

def _ode(self, x, t, k):
y = x
u = self._get_u(x, t)
lin_par_1, lin_par_2, nonlin_par_1, nonlin_par_2 = k

dydt = (lin_par_1 * self.nonlin_fcns[self.nonlin_type](u, nonli
return dydt

def _default_control(x, t):
return 0

def _get_u(self, x, t):
return self._ctrl_fcn(x, t)

def set_u_fcn(self, new_u):
self._ctrl_fcn = new_u

def calcODE(self, y0, ts=800, nt=1001):
y0 = [y0, ]
t = np.linspace(0, ts, nt)
args = (self._params,)
sol = odeint(self._ode, y0, t, args)
return sol, t

def getODE(self):
return self._ode

def get_nonlinear_element_type(self):
return self.nonlin_names[self.nonlin_type]
```

4.2 Свободные движения объекта

В качестве демонстрации и римера работоспособности модели построим свободные движения системы. Свободные движения системы - это движения системы, с подачей нуля на вход.

```
obj = real_object(variant)
y0 = 1
sol, t = obj.calcODE(y0)
plt.plot(t, sol)
plt.grid()
plt.show()
```

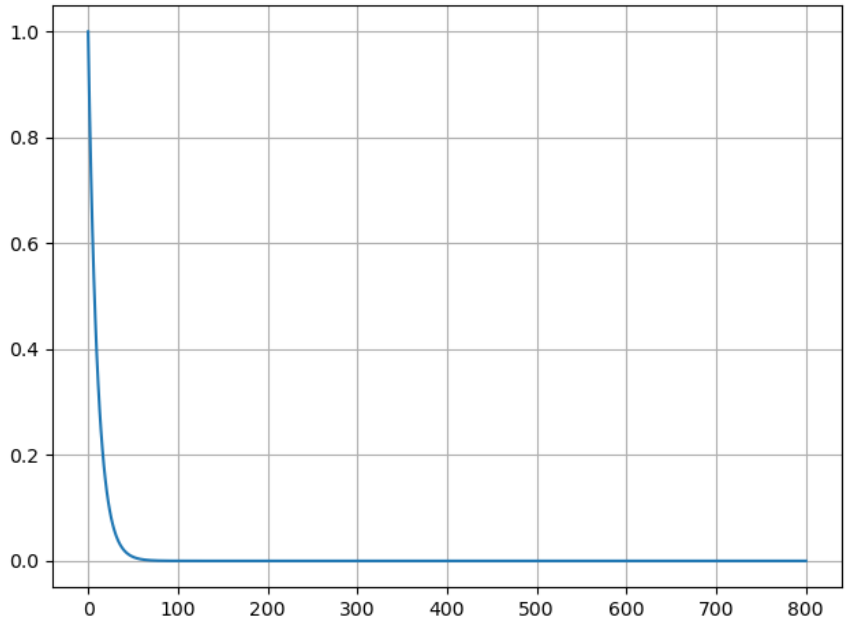

Рисунок 1 — Свободные движения системы

На рисунке 1 продемонстрированы свободные движения системы, результат выше описанного кода.

4.3 Проведение экспериментов

Первым шагом анализа объекта является проведение экспериментов. Одним из основных методов проведения экспериментов является подача на вход объекта тестового сигнала. Минусом такого подхода является потенциальная возможность выхода из строя исследуемого объекта, так что в инженерной практике стоит применять такой подход с осторожностью.

В рамках лабораторной работы будет исследоваться отклик объекта на импульсное, ступенчатое, гармоническое и случайное воздействие. В качестве тестовых воздействий использовать:

- импульсное воздействие длинной 0,01 с и амплитудой 100 (приблилительная аппроксимация дельта-функции);

- единичное ступенчатое воздействие;

- меандр;
- случайное воздействие типа "белый шум"или "окрашенный шум".

Ниже будет описана функция для проведения экспериментов.

```
def exp(sig):
        obj = real_object(variant)

        obj.set_u_fcn(sig)

        y0 = 1

        sol, t = obj.calcODE(y0)

        u = sig(0, t)
        plt.plot(t, u)
        plt.plot(t, sol)
        plt.grid()
        plt.show()
```

На вход подается сигнал, на выходе мы увидим реакцию на гарфики.

4.3.1 Моногармонический сигнал

```
def monoharm_u(x, t):
        return 1*np.sin(t * 0.025 * np.pi)

exp(monoharm_u)
```

Первым шагом в проведение эксперимента является получение сигнала. Положим, что к качестве тестового сигнала будем использовать моногармонический сигнал амплитудой 1 и частотой 0.025 герц.

Так как для задачи моделирования объекта необходимо передавать функцию, а не конкретную реализацию, реализуем функцию, возвращающую моногармонический сигнал и проведем эксперимент. Результат эксперимента будет отображен на рисунке 2

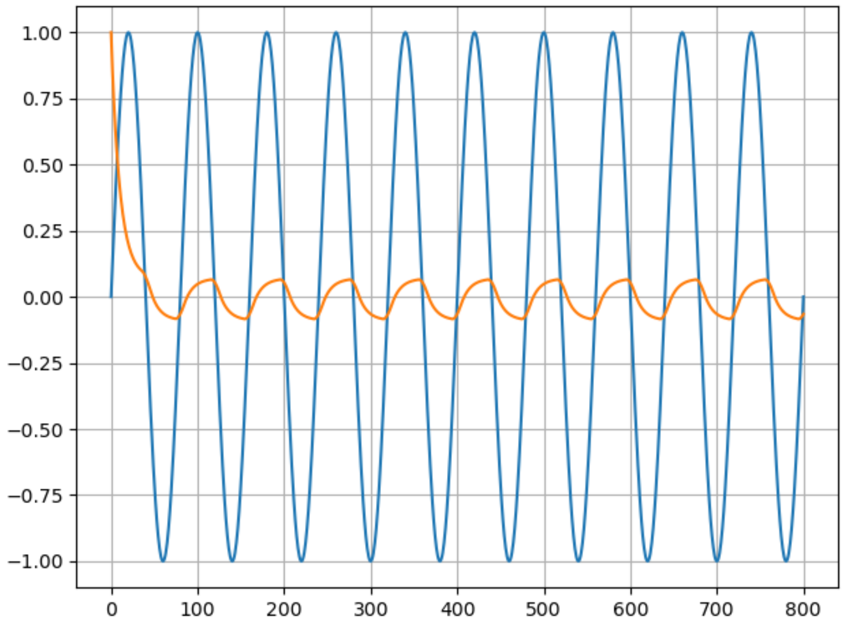

Рисунок 2 — Отклик системы на моногармоническое воздействие

4.3.2 Импульсное воздействие

Рассмотрим воздействие импульсного сигнала длинной 0,01 с и амплитудой 100, что является приблилительной аппроксимацией дельта-функции, на систему. Для этого реализуем функцию на языке Python:

```
def impulse(x,t):
        if t < 0.01:
                return 100
        else:
                return 0
impulse = np.vectorize(impulse,  otypes = [np.float])

exp(impulse)
```

Проведем вычисления и проиллюстрируем получившийся результат на рис. 3:

Можно заметить, что при отсутствие сигнала на входе система равна нулю.

4.3.3 Единичное ступенчатое воздействие

В данной секции рассмотрим воздействие единичной ступенчатой функции. Реализем ее на языке Python:

```
def step_function(x, t):
        if(t >= 0):
                return 1
```

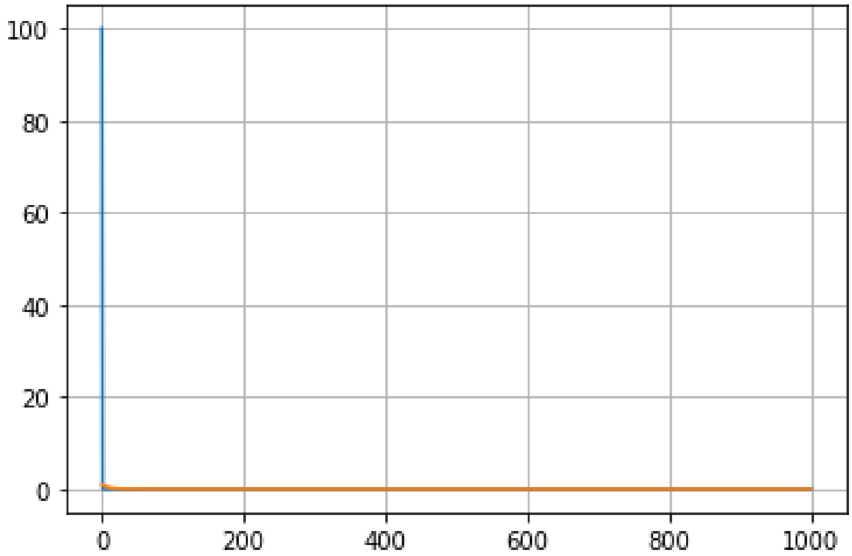

Рисунок 3 — Отклик системы на импульс

```
        else:
                return 0
step_function = np.vectorize(step_function, otypes = [np.float

exp(step_function)
```

Получим значения для системы и продемонтрируем на рис. 4:

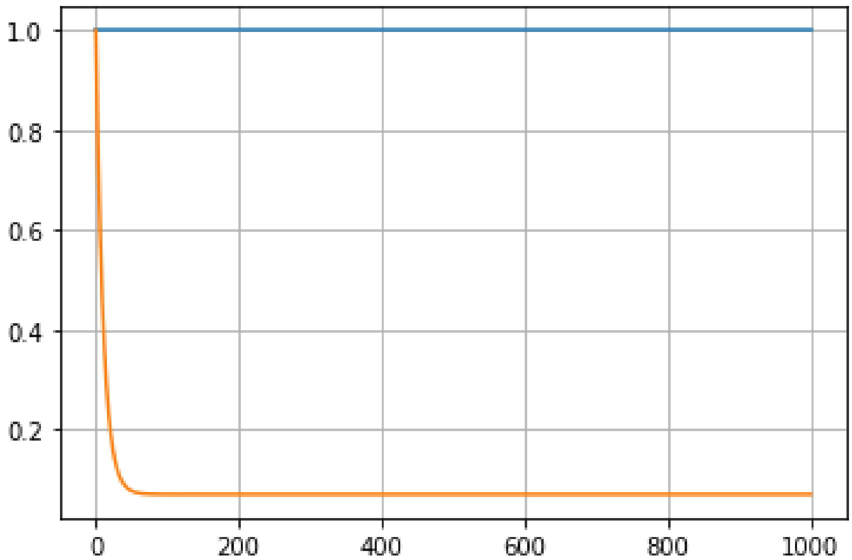

Рисунок 4 — Отклик системы на единичное ступенчатое воздействие

Данный эксперемент показывает, что воздействие формы единичной ступенчатой функции совпадает с сосбственными движениями системы.

4.3.4 Меандр

В данной секции рассмотрим воздействие сигнала вида меандр. Реализем соответствующую функцию на языке Python. Используем так же маетриалы библиотеки scipy/signal для создания меандра.Реализуем в коде:

```
def square(x, t):
        return 1 * signal.square(t * 0.025 * np.pi)

exp(square)
```

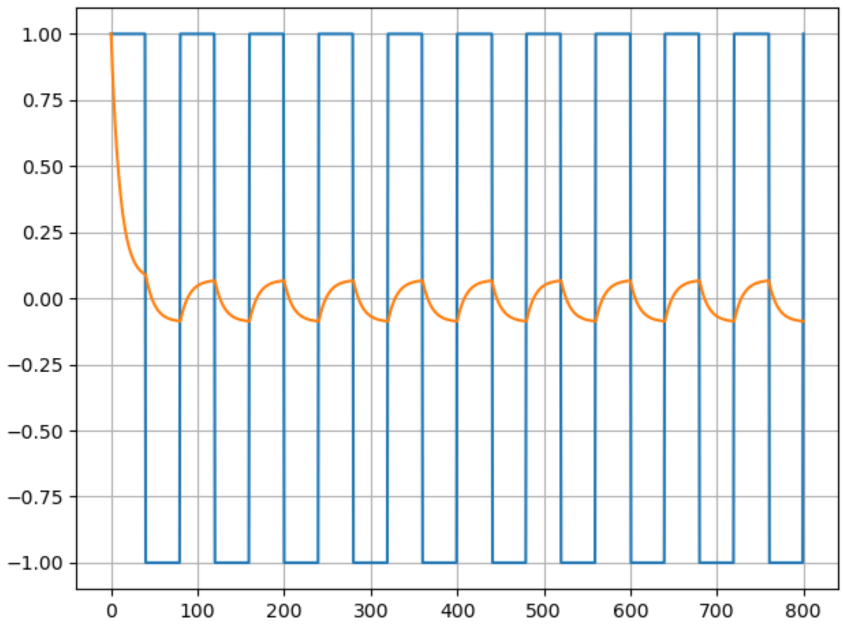

Рисунок 5 — Отклик системы на меандр

Как можно заметить из рисунка 5, сигнал отдаленно напоминает сигнал треугольной формы. Так же определенно видно некоторое сходство с моногоромническим сигналом на рисунке 2.

4.3.5 Случайное воздействие типа "белый шум" или "окрашенный шум"

В данной секции рассмотрим воздействие случайное воздействие на сиситему вида "белый шум". Реализум соответствующую функцию на языке Python, с помощью некоторых библеотечных фукций:

```
def whitenoise(x, t):
        return np.random.normal(1,1)
whitenoise_v = np.vectorize(whitenoise, otypes=[np.float])

exp(whitenoise)
```

Убедимся в корректности результатов, построив график для данной вариации системы (рис. 6):

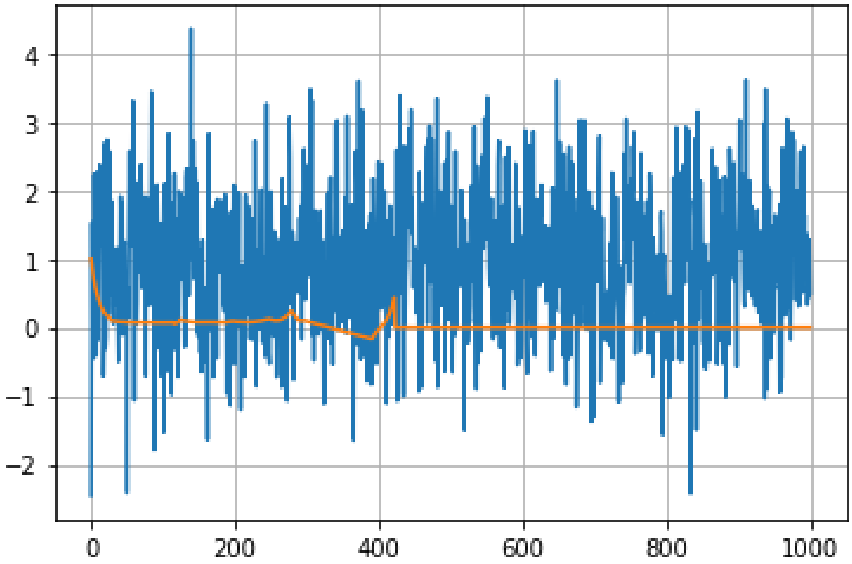

Рисунок 6 — Отклик системы на случайное воздействие

Из результатов видно, что объект подаляет шумовое воздействие: реакция системы имеет низкие амплитудные значения.

4.4 Создание модели и оценка ее параметров

Для оценки параметров подлючим необходимые пакеты и реализуем соответствующий класс:

```
from scipy import optimize
from scipy import integrate, interpolate

class parameter_estimator():
        def __init__(self, experiments, f):
                self._experiments = experiments
                self._f = f
                x_data, y_data = experiments[0]
                self.n_observed = 1 # y_data.shape[1]

        def my_ls_func(self, x, teta):
                r = integrate.odeint(lambda y, t: self._f(y, t
                teta[0:self._y0_len], x)
                return r[:, 0:self.n_observed]

        def estimate_ode(self, y0, guess):
                self._y0_len = len(y0)
```

```
        est_values = np.concatenate((y0, guess))
        c = self.estimate_param(est_values)
        return c[self._y0_len:], c[0:self._y0_len]

    def f_resid(self, p):
        delta = []
        for data in self._experiments:
        x_data, y_data = data
        d = y_data - self.my_ls_func(x_data, p)
        d = d.flatten()
        delta.append(d)
        delta = np.array(delta)
        return delta.flatten()

    def calcODE(self, args, y0, x0=0, xEnd=1000, nt=100001
        t = np.linspace(x0, xEnd, nt)
        sol = odeint(self._f, y0, t, args)
        return sol, t

    def estimate_param(self, guess):
        self._est_values = guess
        res = optimize.least_squares(self.f_resid, sel
        return res.x
```

4.4.1 Идеальная модель

Для демонстрации работоспособности инструмента оценки параметров произведем оценку параметров "идеальной модели". Реализуем на языке Python, так же используем метод класса real object, для получения ДУ :

```
def ideal_model(signal):
    guess = [0.2, 10.25, 0.35, 0.45]
    y0 = [1, ]

    obj = real_object(variant)
    obj.set_u_fcn(signal)
    sol, t = obj.calcODE(y0[0])

    estimator = parameter_estimator([[t, sol], ], obj.getOD
    est_par = estimator.estimate_ode(y0, guess)
    print("Estimated parameter: {}".format(est_par[0]))
    print("Estimated initial condition: {}".format(est_par
```

```
y0 = est_par[1]
args = (est_par[0],)
sol_ideal = odeint(obj.getODE(), y0, t, args)

plt.plot(t, sol_ideal)
plt.plot(t, sol)
plt.grid()
plt.show()
```

Построим гарфики, для всех типов сигналов которые мы использовали для проведения экспериментов.

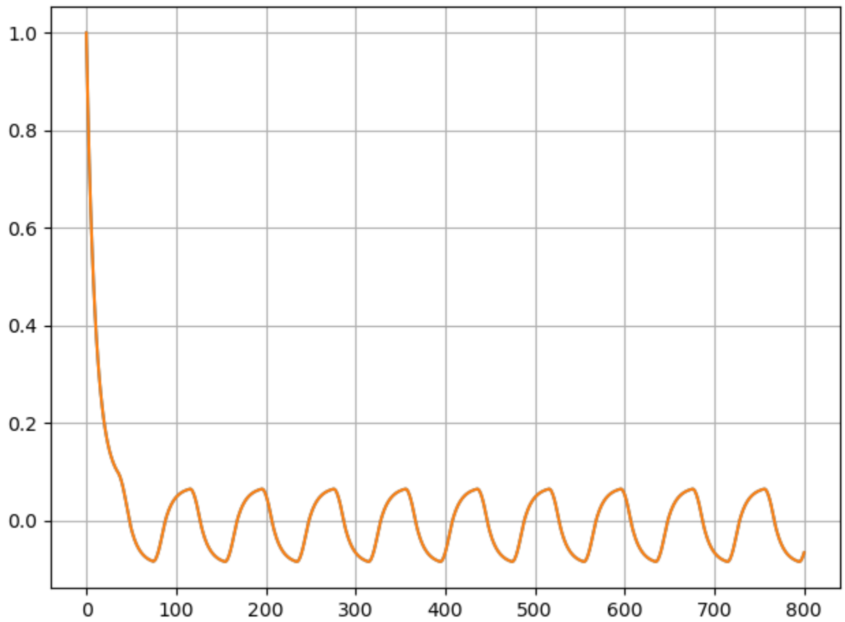

Рисунок 7 — Идеальная модель системы при синусоидальном воздействии

Как видно из рисунка 7, сигнла получилася точной копией того что получился на рисунке 2. Что говорит о том что модель построена верно. Но для чистоты эксперимента проведем эммитации дальше.

Следующим будет сигнал импульс, и его реакция на идеальную модель(см.рисунок 8).

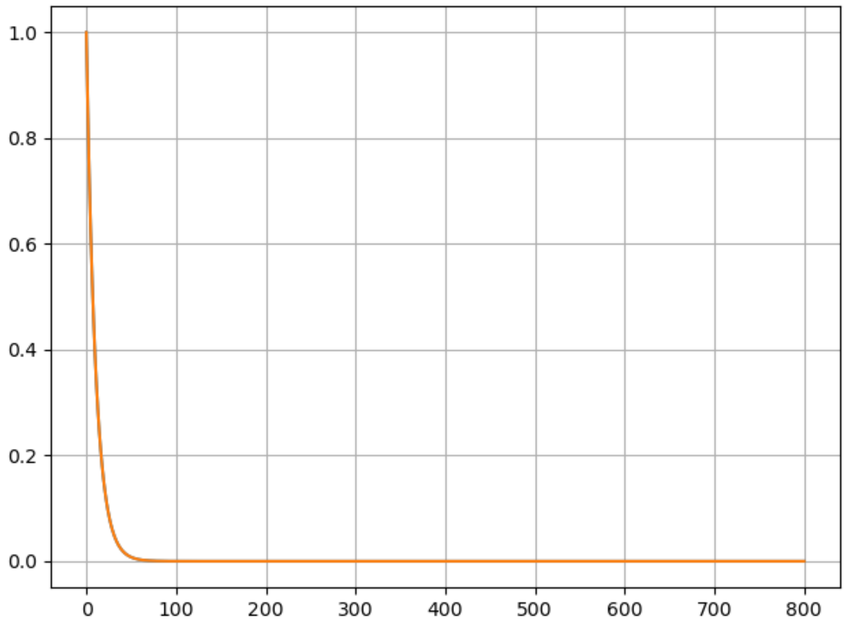

Рисунок 8 — Идеальная модель системы при импульсном воздействии

Аналогичная ситуация показана на рисунке 8.
Слудющим будет ступенчатое воздейтствие.

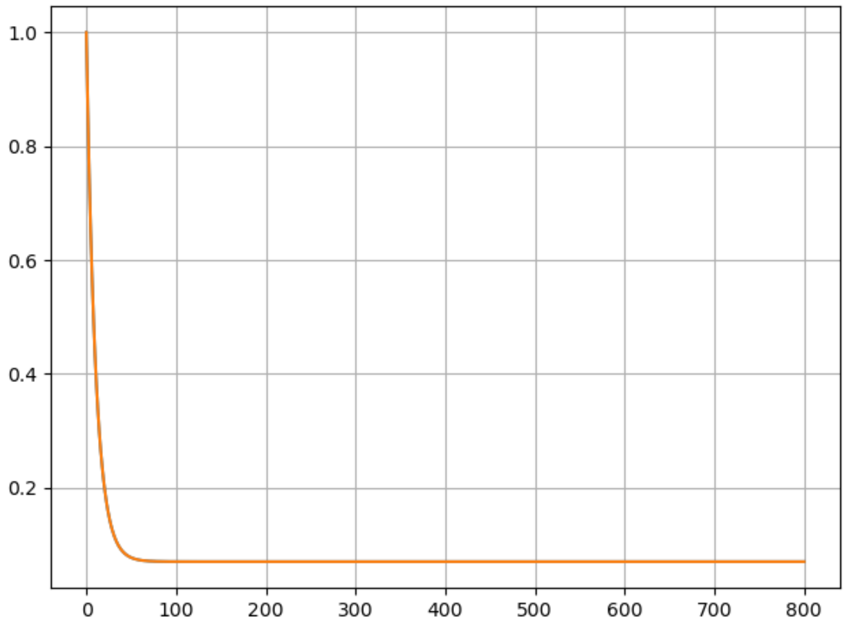

Рисунок 9 — Идеальная модель системы при ступенчатом воздействии

Как видно из рисунка 9, сигнла получилася точной копией того что получился на рисунке 4. Что говорит о том что модель построена верно. Но для чистоты эксперимента проведем эммитации дальше.

Следом идет меандр.

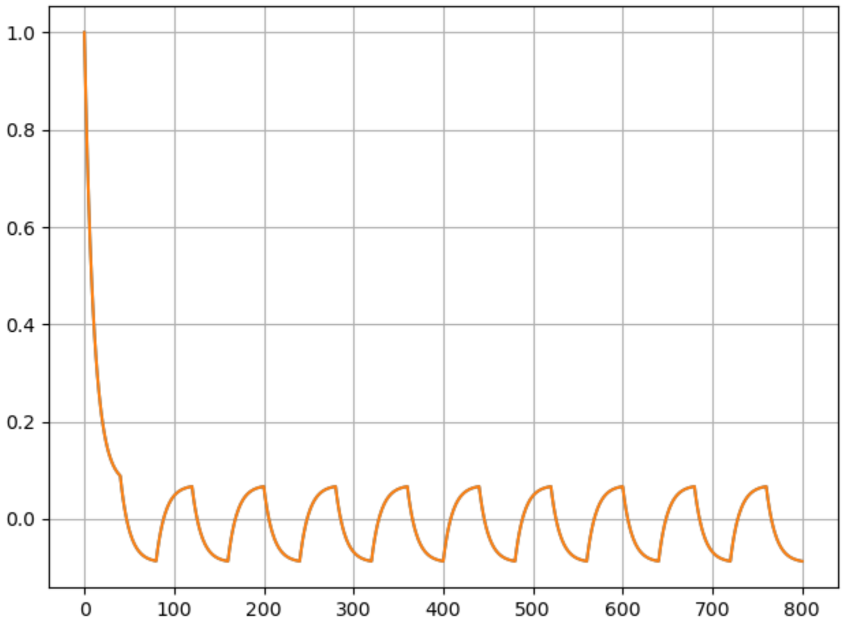

Рисунок 10 — Идеальная модель системы при воздействии сигнала типа "Меандр"

Аналогичная ситуация с меандром, на рисунке10 продемострировано полное сходство с сигналом на реальном объекте.

Теперь плавно подходим к последнем сигналу, это сигнал типа "белый шум"

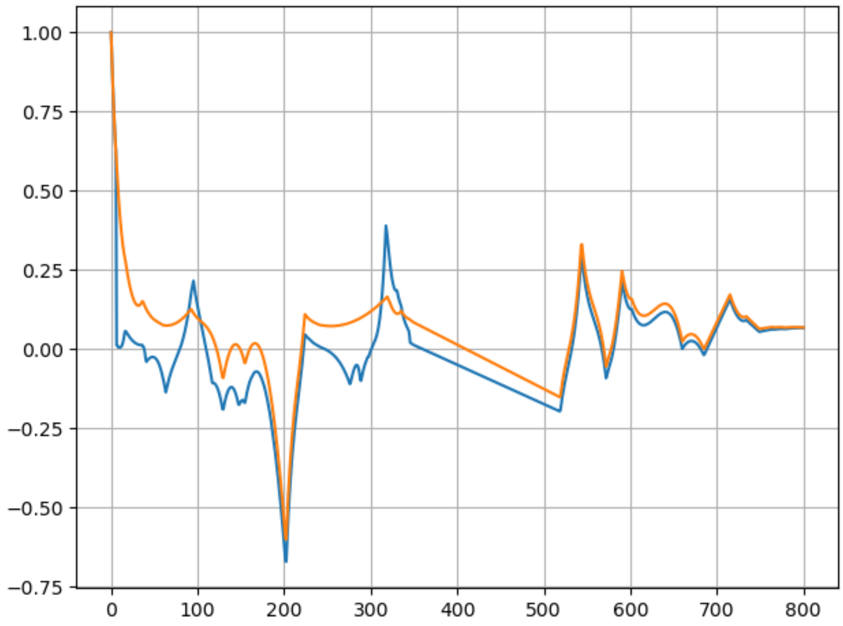

Рисунок 11 — Идеальная модель системы при воздействии сигнала типа "Белый шум"

Как мы видим из графика(см. рисунок 11), что наблюдется некторое расхождение в сигналах, хотя и сигналы отдаленно напоминают друг друга, это не показывает не говорит о том что идеальная модель совпадает с реальное полностью, следовательно есть небольшие вычислительные ошибки, которые обосновываются малой вычислительной мощностью ПК на котором был запущен код.

4.5 Линейная модель

Реализуем функционал для расчета необходимых параметров на языке Python:

```
def ode_lin(x, t, k):
        y = x
        K = k[0]
        T = k[1]
        u = monoharm_u(0, t)
        dydt = (K*u-y)/T
        return dydt

def lineary_model(signal):
        guess = [0.2, 10.25, 0.35, 0.45]
        y0 = [0,]

        obj = real_object(variant)
        obj.set_u_fcn(signal)
        sol, t = obj.calcODE(y0[0])

        estimator = parameter_estimator([[t, sol],], ode_lin)
        est_par = estimator.estimate_ode(y0, guess)
        print("Estimated parameter: {}".format(est_par[0]))
        print("Estimated initial condition: {}".format(est_par

        y0 = est_par[1]
        args = (est_par[0],)
        sol_lin = odeint(ode_lin, y0, t, args)

        plt.plot(t, sol_lin)
        plt.plot(t, sol)
        plt.grid()
        plt.show()
```

Построим графики реакции объекта на все сигналы описанные выше.

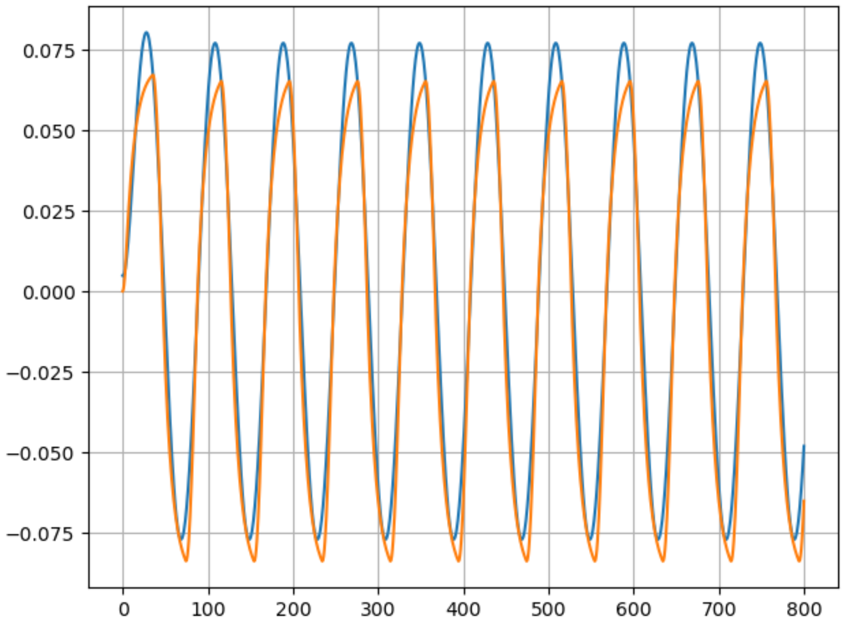

Рисунок 12 — Линейная модель системы при моногармоническом воздействии

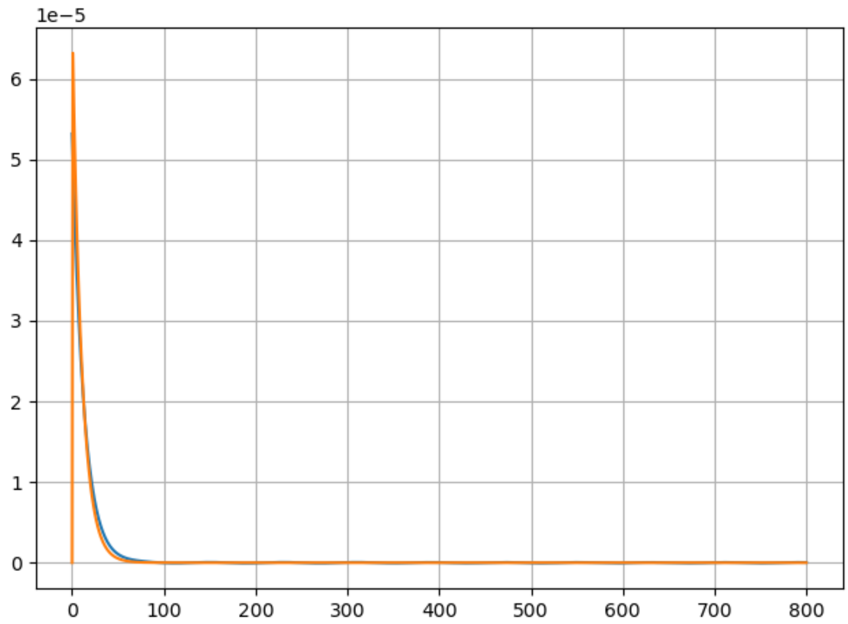

Рисунок 13 — Линейная модель системы при импульсном воздействии

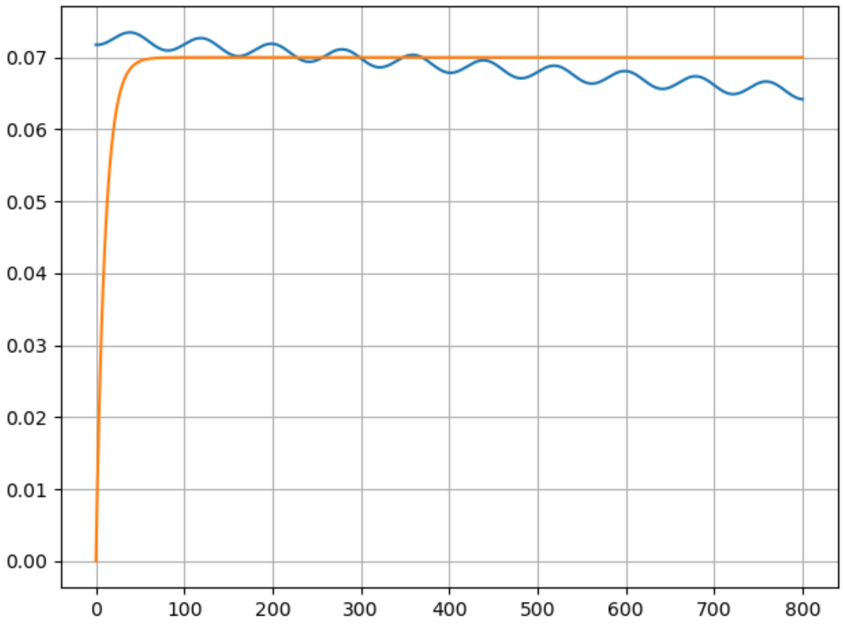

Рисунок 14 — Линейная модель системы при ступенчатом воздействии

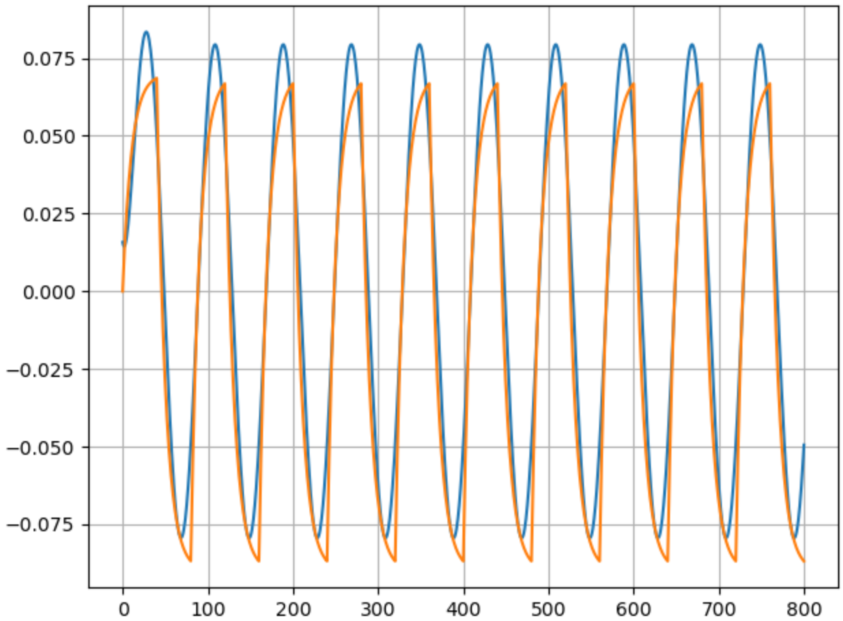

Рисунок 15 — Линейная модель системы при воздействии сигнала типа "Меандр"

Проведя анализ полученных результатов, на выше приведенных графиках, можно с уверенностью сказать что в нашем объекте присутствует нелинейность, так как только одна модлеь парктически соответсвоовала реальном объекту(см.рисунок 13). Особенно хорошо заметно, что в системе присутствует нелинейность, на рисунках 12 и 15, на них можно наблюдать что сигнал старается соблюдать форму, но его "холмы", срезаны, что может подсказать нам на тип нелинейности - насыщение.

4.5.1 Нелинейная модель

Исходя из ранее полученных резульататов, был определен тип нелинейности "Мёртвая зона". В соответствии с данной нелинейностью выполненим подстройку параметров нелинейной модели.

Реализуем модель системы на языке Python:

```
def ode_non_lineary(x, t, k):
        obj = real_object(variant)
        dydt = obj.saturation(x, k[0], k[1])
        return dydt

def non_lineary_model(signal):
        guess = [1, 2]
        y0 = [1, ]

        obj = real_object(variant)
        obj.set_u_fcn(signal)
        sol, t = obj.calcODE(y0[0])

        estimator = parameter_estimator([[t, sol],], ode_non_l
        est_par = estimator.estimate_ode(y0, guess) #Error
        print("Estimated parameter: {}".format(est_par[0]))
        print("Estimated initial condition: {}".format(est_par

        y0 = est_par[1]
        args = (est_par[0],)
        sol_nonlin = odeint(ode_non_lineary, y0, t, args, mxst

        plt.plot(t, sol_nonlin, label = "Nonlinear")
        plt.plot(t, sol, label = "Real")
        plt.grid()
        plt.legend()
        plt.show()
```

Проведем эксперемент на моногармоническом сигнале, пропусти его через наш объект в котором мы учитываем нелинейности типа насыщение.

Как мы можем заметить на рисунке 16 что сигнал практически идельно повторяет форму реального объекта, эта неидеальность заключается в том что возможно мы не совсем корректно подобрали коэффициенты, или же ограничены вычислительными мощностями ПК, на котором выполнялся данный код. Из выше описанного можно сделать вывод что тип нелинейности определен верно.

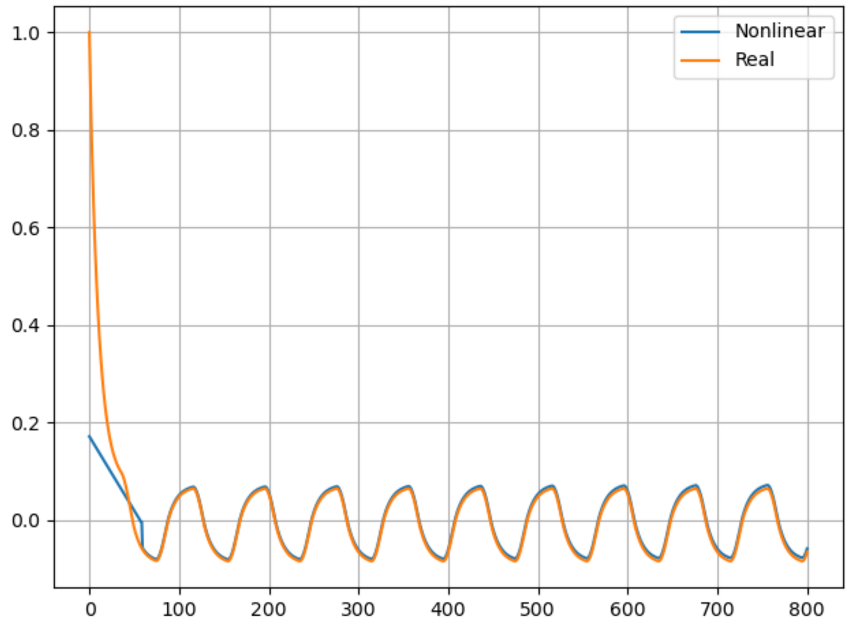

Рисунок 16 — Нелинейная модель при моногармоническом воздействии

4.5.2 Нейросетевая модель

В данной лабораторной работе предлагается воспользоваться библиотекой машинного обучения Keras, которая имеет достаточно простой интерфейс, но содержит большинство современных инструментов в этой области. Нормализуем данные в диапазоне [0, 1] и выберем размер тестовой выборки равный 0.5 от обучающей.

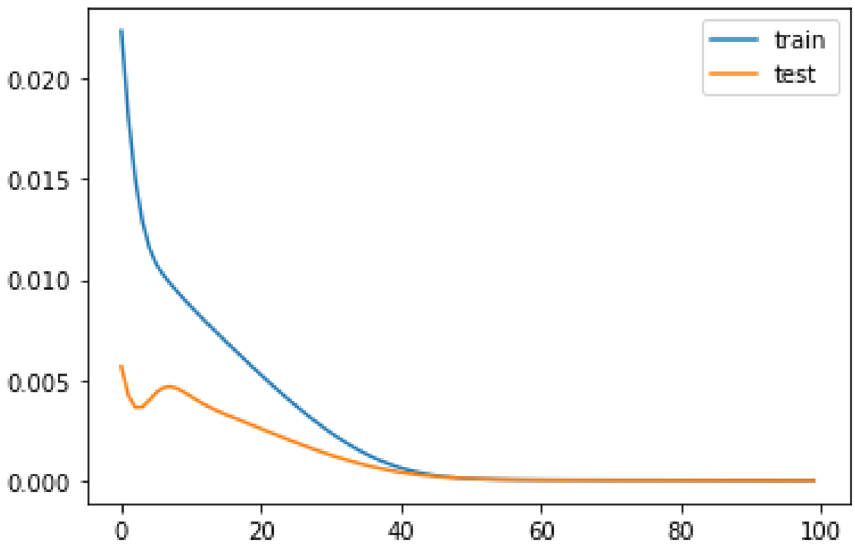

Рисунок 17 — Обучение нейронной сети

После завершения обучения нейронной сети, было построенно предсказание для тестовой выборки и произведен расчёт его среднеквадратичного отклонения.

Среднеквадратичное отклонение : 0.007, полученное значение сведетельствует об успешном обучении нейронной сети.

На рис.18 видно, что прогноз нейронной сети и реальный сигнал практически идентичны друг другу, что подтверждает вышесказанное.

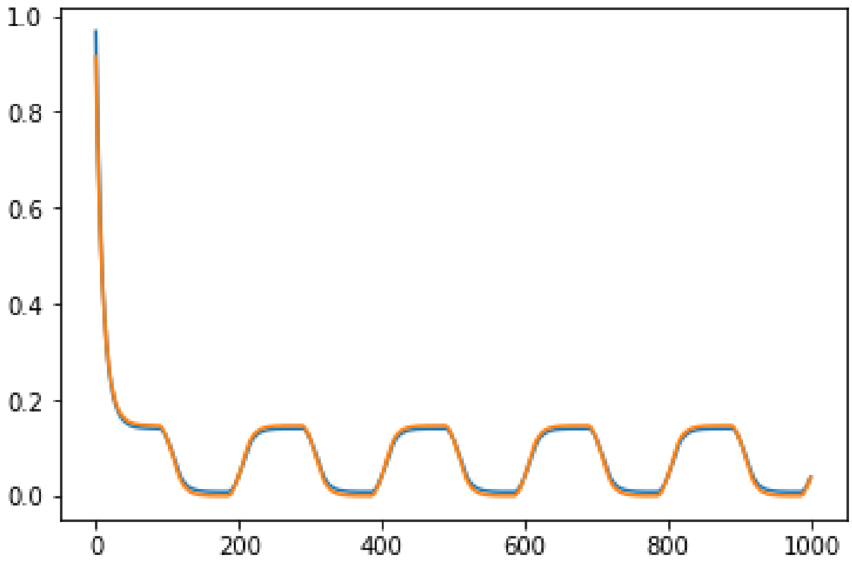

Рисунок 18 — Прогноз нейронной сети

5 ВЫВОДЫ

В лабораторной работе были прводены эксперементы для кмопьютерной идентификации парметров заданного нам параметров управления. С помощью эксперементов мы установили что в объекте есть нелинейность типа насыщения(см.рисунок 16). Так же с помощью функционала языка Python и сторонних библиотек мы собрали датасет из полученных нами результатов, и обучили нейронную сеть на опознования объекта. Так же сделали вывод о том что, когда нет возможности точно узнать(угадать) нелинейность и ее параметры, то хорошим решением может оказаться нейросетевая модель. Был получен хороший результат, среднеквадратичное отклоние равно 0,007(говорит о том, что отклонения незначительны).