ЛР №3 «Вынужденное движение»

Отчет

Студент
Кирилл Лалаянц
R33352
336700
Вариант - 6

Преподаватель
Пашенко А.В.

Факультет Систем Управления и Робототехники

ИТМО

13.09.2023

Содержание

1 Вводные данные

1.1 Цель работы

В этой работе будет проведенно исследование следующих вопросов:

- Вынужденное движение системы в зависимости от корней ее характеристического уравнения.
- Анализ зависимости качества переходных процессов от корней характеристического уравнения системы третьего порядка.
- Проверка свойства свертки как произведения Лапласа.

2 Выполнение работы

2.1 Задание 1. Свободное движение.

2.1.1 Теория

Хотим получить ДУ системы вида:

$$\ddot{y} + a_1\dot{y} + a_0 y = u$$

Сначала получим характеристическое уравнение для частного решения вида:

$$\ddot{y} + a_1\dot{y} + a_0 y = 0$$

Имеем два корня – λ_1 и λ_2. Тогда:

$$(p - \lambda_1)(p - \lambda_2) = p^2 - (\lambda_1 + \lambda_2)p + \lambda_1\lambda_2 = 0,$$

умножив которое на y получаем:

$$\ddot{y} - (\lambda_1 + \lambda_2)\dot{y} + \lambda_1\lambda_2 y = 0.$$

Для данного ДУ известно частичное решение:

$$y_0(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t},$$

параметры c_1 и c_2 которого вычисляются из начальных условий.
Также подставим характеристическое уравнение в знаменатель TF $W(p)$:

$$W(p) = \frac{1}{p^2 - (\lambda_1 + \lambda_2)p + \lambda_1\lambda_2}$$

Полученная TF соотвествует системе с желаемыми корнями характеристического уравнения.

2.1.2 Программная реализация

```
def task1_output(m1, m2, initial_values, us, ts, plot_name, save_name):
    poly = sympy.simplify((p - m1) * (p - m2))
    coeffs = sympy.Poly(poly, p).all_coeffs()
    ss = control.tf2ss(control.tf(1, np.array(coeffs, dtype=np.float64)))
    ss_reachable = control.canonical_form(ss, form="reachable")[0]
    responses = {}
    for key_u, u in us.items():
        responses[key_u] = []
        for initial_value in initial_values:
            response = control.forced_response(ss_reachable, U=u(ts), X0=
                                                initial_value, T=ts)
            responses[key_u].append((initial_value, response))

    plot_task1(responses, ts, plot_name, save_name)
```

2.1.3 Результаты

Сначала было проведено исследования системы с двумя вещественными корнями.

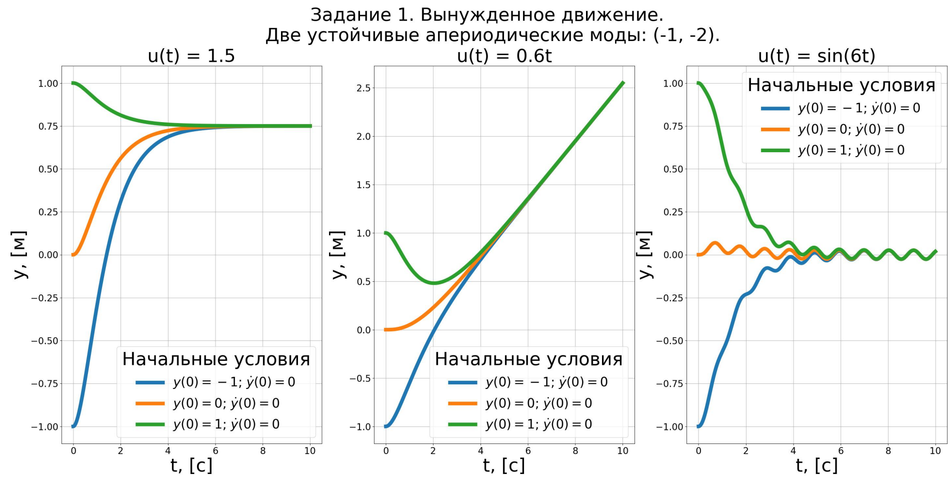

Рис. 1: Результат двух устойчивых апереодических мод.

По графигам на рисунке 1 видно, что устойчивая система при подаче константного воздействия – стабилизируется; при неограниченно растущем – уходит в бесконечность; при периодическом – коллеблется.

Исследование системы с нейтральным и устойчивым корнем дало следующие результаты. Видно, что системы стабильна только при переодическом воздействии.

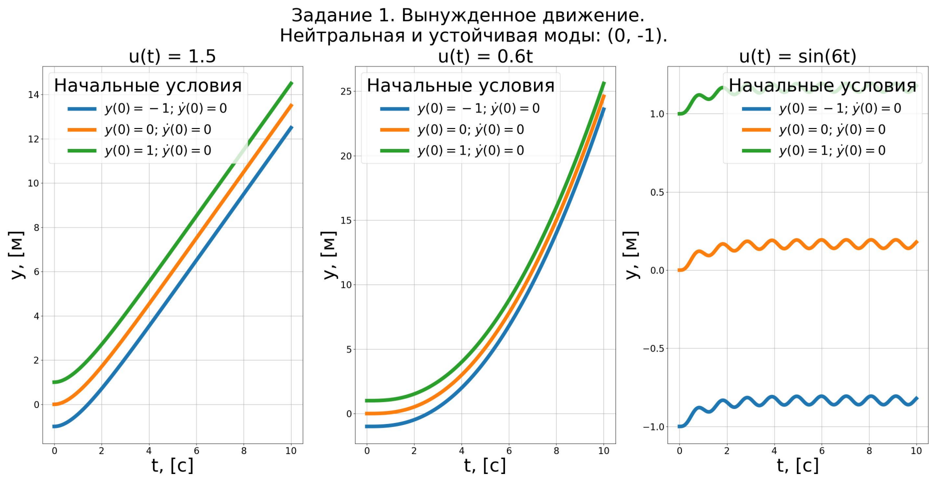

Рис. 2: Результат нейтральной и устойчивой апереодических мод.

Такое поведение легко объясняется решением ДУ аналитически.

Последняя исследованная система имела два неустойчивых колебательных корня. Результат, ожидаемо, крайне неустойчив.

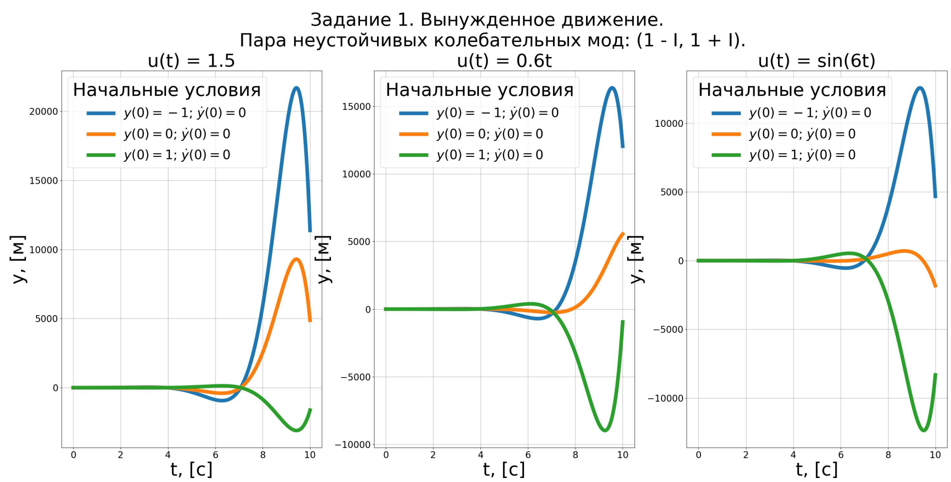

Рис. 3: Результат двух неустойчивых колебательных мод.

2.2 Задание 2. Качество переходных процессов.

2.2.1 Теория

В этом задании TF получается аналогично первому заданию. После этого определяется два параметра для оценки переходного процесса:

- $T_{5\%} : \forall t > T_{5\%} \hookrightarrow \frac{|y(t)-y(T_{5\%})|}{y(T_{5\%})} < 0.05$ – время переходного процесса;
- $\Delta\sigma = |\frac{y_{max}-y_{\infty}}{y_{\infty}}|$ – перерегулирование;

2.2.2 Программная реализация

```
    y = sympy.Function("y")
t = sympy.Symbol("t")

def get_ss_reachable(modes):
    poly = 1
    for mode in modes:
        poly = sympy.simplify(poly * (p - mode))
    coeffs = sympy.Poly(poly, p).all_coeffs()

    ss = control.tf2ss(control.tf(1, np.array(coeffs, dtype=np.
                                                float64)))
    ss_reachable = control.canonical_form(ss, form="reachable")[0]
    return ss_reachable



def get_state_limit(ss_reachable, T0, initial_values_T0):
    params = np.concatenate((ss_reachable.A[0, :], ss_reachable.B[0,
                                                :]))
    a2, a1, a0, b0 = map(float, -params)
    b0 = -b0

    ics={y(T0): initial_values_T0[0], y(t).diff(t, 1).subs(t, T0):
                                                initial_values_T0[1], y(t).
                                                diff(t, 2).subs(t, T0):
                                                initial_values_T0[2]}
    ode_sympy = sympy.dsolve(y(t).diff(t, 3) + a2 * y(t).diff(t, 2) +
                                                 a1 * y(t).diff(t, 1) + a0*
                                                y(t) - 1, X0=0, ics=ics)
    time_of_limit = 10**10
    state_limit = ode_sympy.subs(t, time_of_limit).rhs
    while abs(1 - state_limit/ode_sympy.subs(t, time_of_limit * 10).
                                                rhs) > 0.001:
```

```
        time_of_limit *= 10
        state_limit = ode_sympy.subs(t, time_of_limit).rhs
    return state_limit

def model_until_5percent_band(ss_reachable, initial_values,
                                  state_limit):
    percent5_interval = state_limit * 0.05
    t_max = 10
    while True:
        ts = get_t(t_max)
        response = control.forced_response(ss_reachable, U=1, X0=
                                          initial_values[::-1], T
                                          =ts)

        t_5_percent = 0
        for i in range(len(ts)):
            if abs(response.outputs[i] - state_limit) >
                                          percent5_interval:
                t_5_percent = ts[i]
                t_5_percent_i = i
                y_5_percent = response.outputs[i]
        if t_5_percent <= t_max*0.8:
            ts = get_t(t_max)

            response = control.forced_response(ss_reachable, U=1, X0=
                                              initial_values[::-1
                                              ], T=ts)

            return (ts, response.outputs, t_5_percent, y_5_percent,
                                              t_5_percent_i * 2)
        t_max *= 1.5

def get_overshooting(response_outputs, ts, state_limit):
    overshooting_values = response_outputs - state_limit
    if overshooting_values[0] > 0:
        overshooting_values *= -1
    overshooting_counter = 0
    while overshooting_values[overshooting_counter] <
                                      overshooting_values[
                                      overshooting_counter + 1]:
        overshooting_counter += 1
        if overshooting_counter >= len(overshooting_values) - 1:
            break
```

```
    absolute_overshooting = overshooting_values[overshooting_counter]
    relative_overshooting = overshooting_values[overshooting_counter]
                                              / state_limit
    y_overshooting = response_outputs[overshooting_counter]
    t_overshooting = ts[overshooting_counter]
    return (t_overshooting, y_overshooting, relative_overshooting,
                                             absolute_overshooting)



def task2_analyse(modes, T0 = 0, initial_values_T0 = [0, 0, 0]):
    results = {}
    for title, mode in modes.items():
        ss_reachable = get_ss_reachable(mode)
        ss_limit = get_state_limit(ss_reachable, T0,
                                                initial_values_T0)
        ts, ss_y, t_5p, y_5p, _ = model_until_5percent_band(
                                                ss_reachable,
                                                initial_values_T0,
                                                ss_limit)
        t_os, y_os, os_rel, _ = get_overshooting(ss_y, ts, ss_limit)
        results[title] = [mode, ts, ss_y, ss_limit, t_5p, y_5p, t_os,
                                                 y_os, os_rel]
    return results
```

2.2.3 Результаты

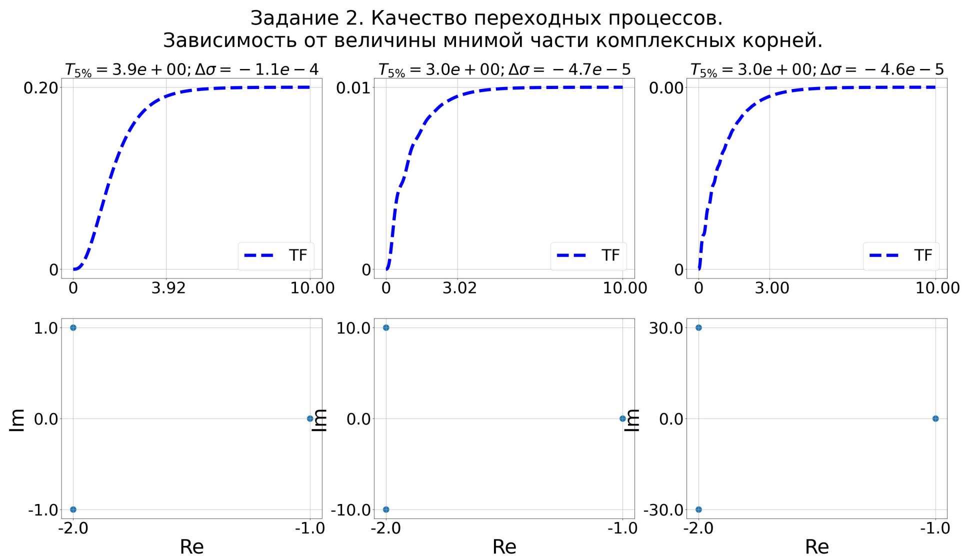

Рис. 4: Зависимость от мнимой части комплексных корней.

Сначала было проведенно исследование зависимости от мнимой части комплексных корней (рис. 4). Можно заметить, что при ее уменьшении, время переходного процесса и перерегулирование падают.

Затем было проведено исследование зависимости от вещественной части мнимых корней (рис. 5). Результат ожидаем – чем меньше вещественная часть, тем быстрее сходимость и меньше перерегулирование.

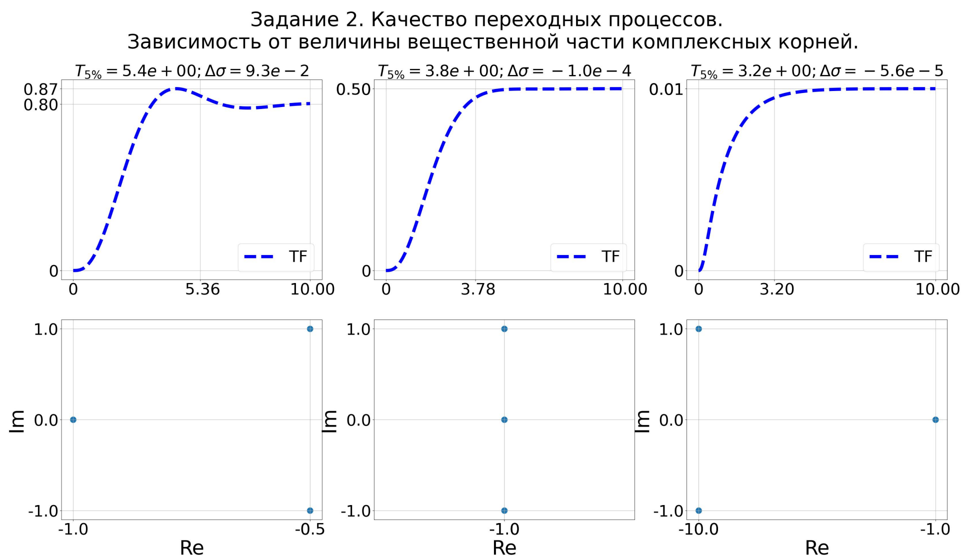

Рис. 5: Зависимость от вещественной части комплексных корней.

Последний эксперимент был нацелен на исследование чисто вещественных корней. Результат (рис. 6) вновь оказался предсказуем: чем меньше вещественная часть – тем быстрее.

В целом можно заметить, что время переходного процесса и величина перерегулирования во многом зависят от самого близкого к нулю по вещественной оси корня. Все остальные корни оказывают меньшее влияние.

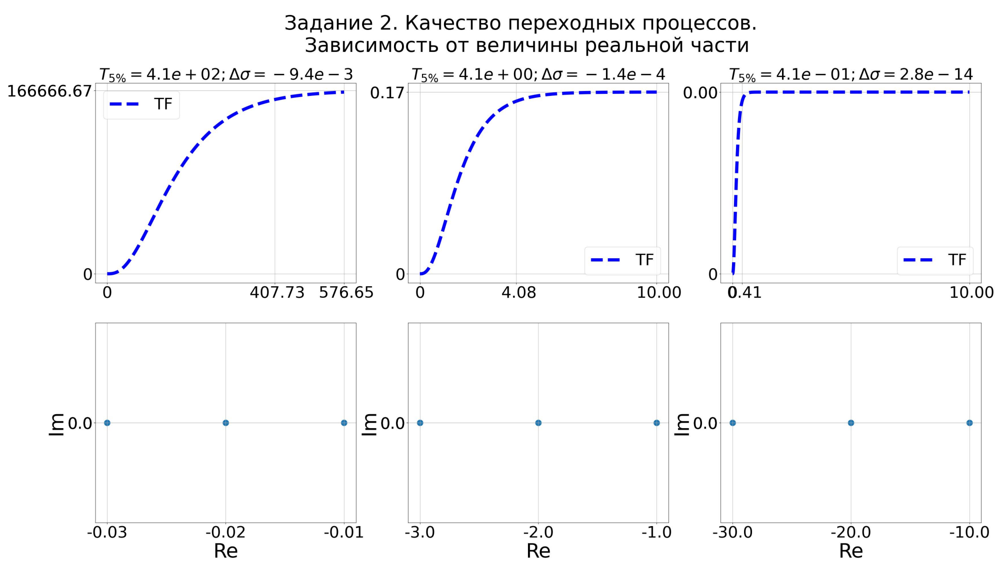

Рис. 6: Зависимость от вещественных корней.

2.3 Задание 3. (Необязательное) Свертка, как произведение образов Лапласа.

2.3.1 Теория

В этом задании проверим свойство свертки как преобразования Лапласа. Имеем систему:

$$W(s) = \frac{6}{(s+2)^4}, u(t) = 1.5 + 0.6t + sin(6t)$$

Далее можем работать с этой системой тремя способами:

1. $y_{i.r.} = \mathscr{L}^-1\{W\} \hookrightarrow y = (y_{i.r.} * u)(t)$ – по свойству свертки;
2. $y(t) = W[u(t)]$ – классическое моделирование системы через TF;
3. $U = \mathscr{L}\{u(t)\} \hookrightarrow y(t) = (WU)[\delta(0)]$ – по свойству перемножения образов Лапласа;

2.3.2 Программная реализация

```
dt = 0.001
ts = get_t(10, dt=0.001)
u_f = lambda t: 4 * np.cos(2*t) +  0.5 * t
u_sympy = 4 * sympy.cos(2*t) + 0.5 * t
u_lambda = sympy.lambdify(t, u_sympy, 'numpy')
U_sympy = sympy.laplace_transform(u_sympy, t, s)[0]
W_s_denum_coeffs = sympy.Poly((s+2)**4, s).all_coeffs()
W_s_denum_coeffs = list(map(float, W_s_denum_coeffs))
W_sympy = 6 / sympy.Poly((s+2)**4, s)
W_control = control.tf([6], W_s_denum_coeffs)
#  Convolutin
y_ir_sympy = sympy.inverse_laplace_transform(W_sympy, s, t)
y_ir_lambda = sympy.lambdify(t, y_ir_sympy, 'numpy')
y_full_convolution = np.convolve(y_ir_lambda(ts), u_f(ts)) * dt

#  Forced response
y_2 = control.forced_response(W_control, T=ts, U=u_lambda(ts)).
                                       outputs

#  Multiplication of Laplace images
U_sympy = U_sympy.simplify()
Y_sympy = U_sympy * W_sympy
TF = Y_sympy.simplify()
num_3 = list(map(float,sympy.Poly(sympy.fraction(TF)[0],s).all_coeffs
                                       ()))
```

```
denum_3 = list(map(float,sympy.Poly(sympy.fraction(TF)[1],s).
                                    all_coeffs()))
tf_3_2=control.tf(num_3, denum_3)
y_3 = control.impulse_response(tf_3_2, T=ts).outputs
```

2.3.3 Результаты

Как видно по рисунку 7, результаты совпали. В очередной раз не получилось подловить математиков на обмане.

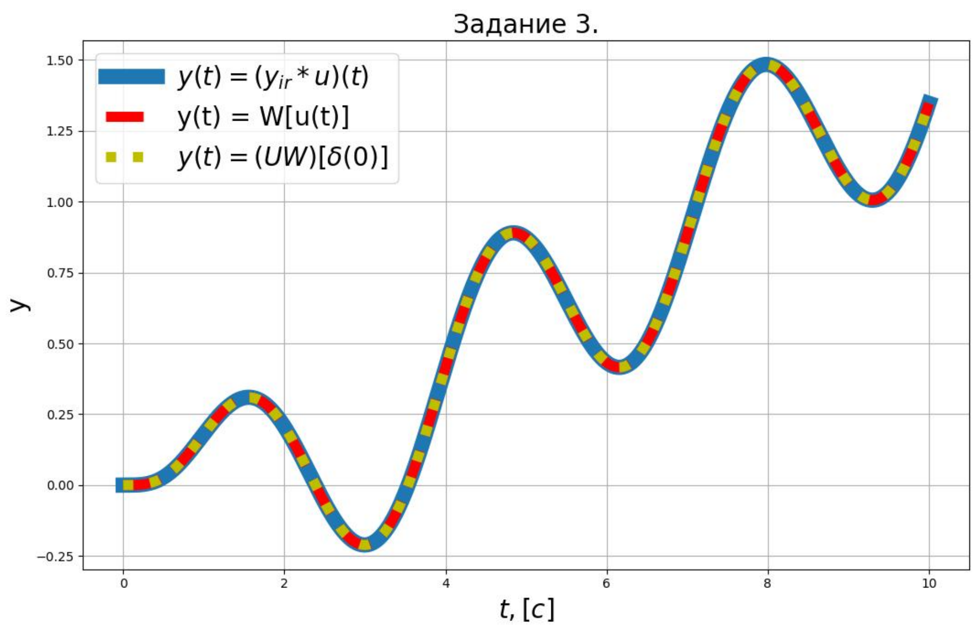

Рис. 7: Сравнение различных способов.

3 Заключение

В этой работе было проведенно исследование следующих вопросов:

- Вынужденное движение системы в зависимости от корней ее характеристического уравнения.
- Анализ зависимости качества переходных процессов от корней характеристического уравнения системы третьего порядка.
- Проверка свойства свертки оригиналов преобразования Лапласа.

3.1 Выводы

1. Проведено моделирование вынужденного движение систем с различным ненулевым входным воздействием.
2. На практике проверенно влияние мод на характер поведения системы.
3. Наглядно проверенно, что свертка оригиналов равносильна перемножению образов Лапласа.