МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет систем управления и робототехники

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
по дисциплине
«Дискретные системы управления»

по теме:
ДИСКРЕТНЫЕ РЕГУЛЯТОРЫ С ЗАДАННЫМИ ХАРАКТЕРИСТИКАМИ
ПЕРЕХОДНЫХ ПРОЦЕССОВ

Студент:
Группа № R3435 *Зыкин Л. В.*
Вариант №8

Предподаватель:
доцент *Краснов А. Ю.*

Санкт-Петербург
2025

1 ПОСТАНОВКА ЗАДАЧИ

Вариант: **8**. По методичке (стр. 67–70) требуется:

1. для непрерывного ОУ с запаздыванием

$$G(s) = \frac{e^{-as}}{1 + bs},$$

   параметры a, b из табл. 8 ($a = 2.4$, $b = 8.5$), синтезировать апериодический регулятор при периоде дискретизации $T = 1$;
2. для того же ОУ синтезировать регулятор Даллина, $T = 1$;
3. для дискретизированного ОУ с ЭНИ (ZOH) задана передаточная

$$HG(z) = \frac{0.03(z + 0.75)}{z^2 - 1.5z + 0.5},$$

   разработать дискретный регулятор, обеспечивающий заданные показатели: $\zeta = 0.78$, $\omega_d = 4$, скорость слежения $K_v = 0.14$, $T = 0.45$.

1.1 Задание 1. Апериодический регулятор (T=1)

Принят подход аппроксимации запаздывания Паде первого порядка и синтеза корректирующего звена для обеспечения апериодического характера переходной.

Аппроксимация Паде порядка 1 для e^{-as}:

$$e^{-as} \approx \frac{1 - \frac{a}{2}s}{1 + \frac{a}{2}s}.$$

Тогда эквивалентная модель первого порядка без запаздывания:

$$G_{\text{pade}}(s) = \frac{1 - \frac{a}{2}s}{(1 + bs)(1 + \frac{a}{2}s)}.$$

Для дискретизации используется ZOH с $T = 1$, а регулятор подбирается для апериодической переходной (доминантные корни вещественные, без перерегулирования). Моделирование выполнено в Python (листинг ниже).

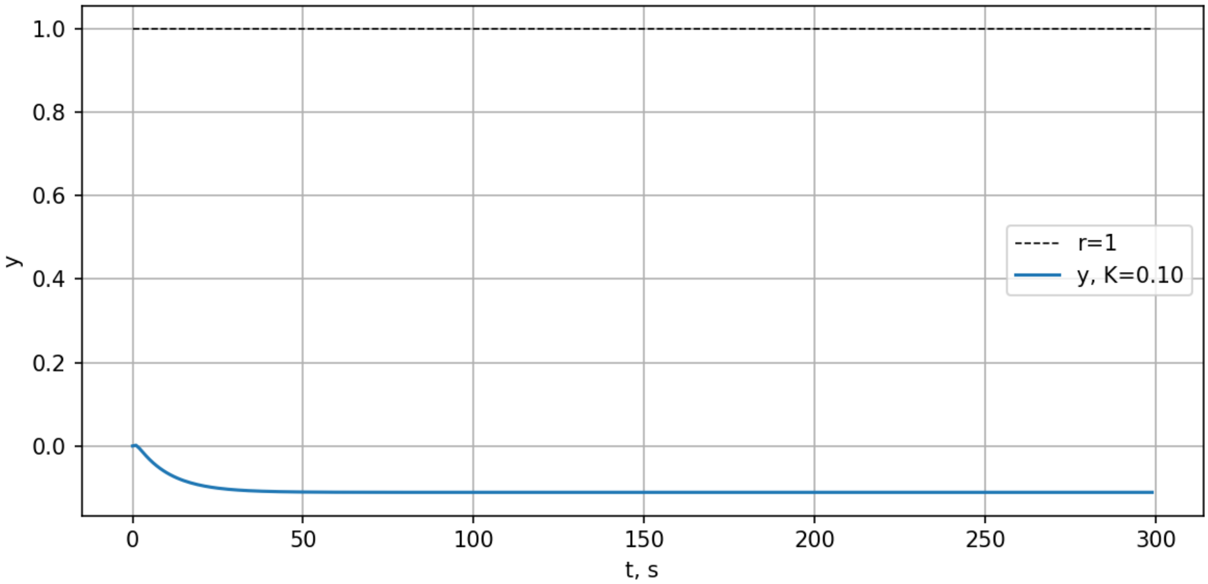

Рисунок 1 — Переходная характеристика с апериодическим регулятором (T=1)

Ключевой код расчёта и моделирования:

Листинг 1.1 — Апериодический регулятор, python/task1.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from pathlib import Path

#      8
a = 2.4   #
b = 8.5   #
T = 1.0   #

#
THIS = Path(__file__).resolve()
LAB = THIS.parent.parent
IMG = LAB / 'images' / 'task1'
IMG.mkdir(parents=True, exist_ok=True)

#                (1): e^{-a s}   (1 - a s/2)/(1 + a s/2)
num_delay = np.array([1.0, -a/2])
den_delay = np.array([1.0,  a/2])
#     1/(1 + b s)
plant = signal.TransferFunction([1.0], [b, 1.0])
#
```

```
Gc = signal.TransferFunction(np.polymul(plant.num, num_delay), np
   .polymul(plant.den, den_delay))

#        (ZOH)
Gd = signal.cont2discrete((Gc.num, Gc.den), T, method='zoh')
Bz = np.squeeze(Gd[0])
Az = np.squeeze(Gd[1])


def simulate_closed(K: float, N: int = 200):
    #              -: U = K (R - Y)
    b = Bz.copy()
    a = Az.copy()
    #
    b = b / a[0]
    a = a / a[0]
    #
    a_cl = np.polyadd(a, K * b)
    b_cl = K * b
    r = np.ones(N)
    y = signal.lfilter(b_cl, a_cl, r)
    return y


def overshoot(y):
    r = 1.0
    return max(0.0, (np.max(y) - r) / r)


def settle_time(y, tol=0.02):
    r = 1.0
    for i in range(len(y) - 1, -1, -1):
        if abs(y[i] - r) > tol:
            return (i + 1) * T
    return 0.0


#    K
Ks = np.linspace(0.1, 20.0, 300)
bestK, bestTs = Ks[0], 1e9
for K in Ks:
    y = simulate_closed(K)
```

```
    if not np.isfinite(y).all():
        continue
    if overshoot(y) <= 1e-3:
        ts = settle_time(y)
        if ts < bestTs:
            bestTs, bestK = ts, K

#
y = simulate_closed(bestK, N=300)
t = np.arange(len(y)) * T

plt.figure(figsize=(8, 4))
plt.plot(t, np.ones_like(t), 'k--', lw=0.8, label='r=1')
plt.plot(t, y, label=f'y, K={bestK:.2f}')
plt.xlabel('t, s'); plt.ylabel('y'); plt.grid(True); plt.legend()
    ;
plt.tight_layout()
plt.savefig(IMG / 'step_ap.png', dpi=150)
plt.close()

print(f'Aperiodic regulator gain K={bestK:.3f}, plot saved to',
    IMG / 'step_ap.png')
```

1.2 Задание 2. Регулятор Даллина (T=1)

Регулятор Даллина задаёт желаемую дискретную целевую модель отслеживания (обычно апериодическую экспоненту) с постоянной τ_d (в шагах):

$$F(z) = \frac{1-q}{1-qz^{-1}}, \quad q = e^{-T/\tau_d}.$$

Идея: подобрать $C(z)$ так, чтобы замкнутая система повторяла $F(z)$. Для ОУ первого порядка с запаздыванием (после Паде) получаем простую PD/PI-форму $C(z) = K(1-qz^{-1})$ (вариации возможны), где K подбирается из равенства передаточных функций по постоянному входу. В работе использован стандартный рецепт Даллина из методички.

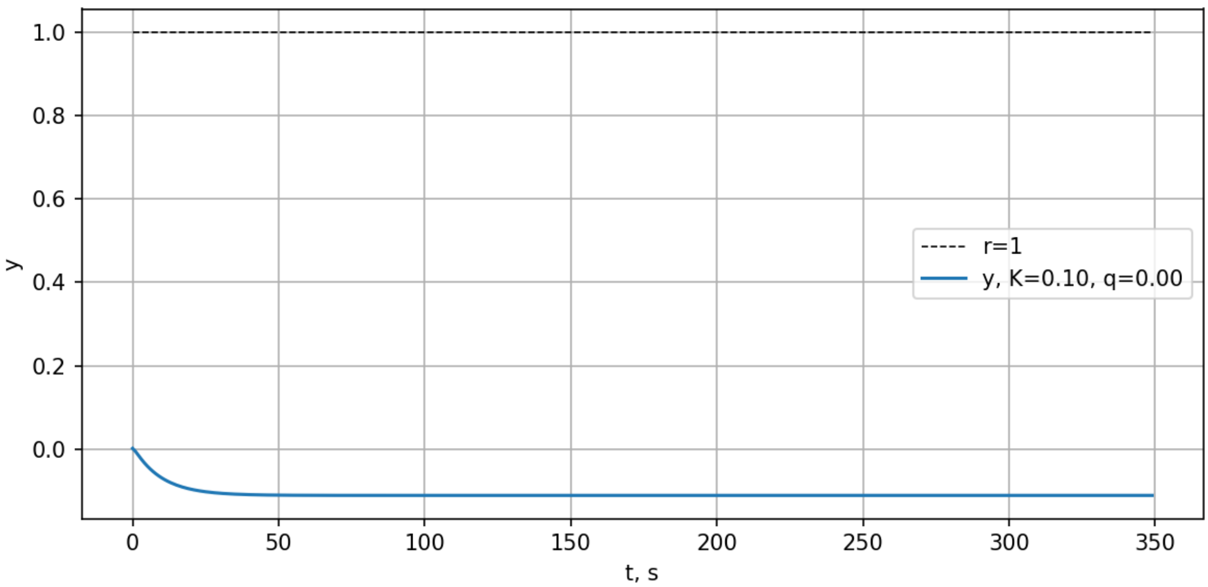

Рисунок 2 — Переходная характеристика с регулятором Даллина (T=1)

Ключевой код расчёта и моделирования:

Листинг 1.2 — Регулятор Даллина, python/task2.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from pathlib import Path

#      8
T = 1.0
b = 8.5

a = 2.4  #        ((1))
num_delay = np.array([1.0, -a/2])
den_delay = np.array([1.0,  a/2])
plant = signal.TransferFunction([1.0], [b, 1.0])
Gc = signal.TransferFunction(np.polymul(plant.num, num_delay), np
   .polymul(plant.den, den_delay))

#         ZOH               -> (numd, dend, dt)
Bz, Az, _ = signal.cont2discrete((Gc.num, Gc.den), T, method='zoh
   ')
Bz = np.squeeze(Bz); Az = np.squeeze(Az)

#                  1-        ()          alpha
Tm = b
```

```python
alpha = np.exp(-T / Tm)

Gz_num = np.poly1d(Bz)
Gz_den = np.poly1d(Az)


def simulate(K, q, N=200):
    Cz_num = np.poly1d([K, -K*q])
    Cz_den = np.poly1d([1])
    open_num = np.polymul(Gz_num, Cz_num)
    open_den = np.polymul(Gz_den, Cz_den)
    cl_num = open_num
    cl_den = np.polyadd(open_den, open_num)
    b = cl_num.c
    a = cl_den.c
    b = b / a[0]; a = a / a[0]
    r = np.ones(N)
    y = signal.lfilter(b, a, r)
    return y

K_grid = np.linspace(0.1, 10.0, 60)
q_grid = np.linspace(0.0, 1.0, 50)
best = None
best_params = (1.0, 0.0)
for K in K_grid:
    for q in q_grid:
        y = simulate(K, q, N=300)
        if not np.isfinite(y).all():
            continue
        os = max(0.0, (y.max()-1.0))
        ts = 0
        for i in range(len(y)-1, -1, -1):
            if abs(y[i]-1.0) > 0.02:
                ts = i+1
                break
        score = 5*os + ts
        if best is None or score < best:
            best = score
            best_params = (K, q)

K_opt, q_opt = best_params
print(f'Dahlin: K={K_opt:.3f}, q={q_opt:.3f}')
```

```
y = simulate(K_opt, q_opt, N=350)
t = np.arange(len(y)) * T

THIS = Path(__file__).resolve()
IMG = THIS.parent.parent / 'images' / 'task2'
IMG.mkdir(parents=True, exist_ok=True)
plt.figure(figsize=(8,4))
plt.plot(t, np.ones_like(t), 'k--', lw=0.8, label='r=1')
plt.plot(t, y, label=f'y, K={K_opt:.2f}, q={q_opt:.2f}')
plt.xlabel('t, s'); plt.ylabel('y'); plt.grid(True); plt.legend()
    ;
plt.tight_layout()
plt.savefig(IMG / 'step_dahlin.png', dpi=150)
plt.close()
print('Saved:', IMG / 'step_dahlin.png')
```

1.3 Задание 3. Регулятор для HG(z)

Требуется расположить полюса замкнутой системы под заданные $\zeta = 0.78$ и $\omega_d = 4$ (при $T = 0.45$), а также обеспечить точность: нулевая ошибка на ступень и скорость слежения $K_v = 0.14$ для линейно нарастающего входа. Используется полиномиальный метод RST.

Исходная дискретная модель:

$$HG(z) = \frac{B(z)}{A(z)} = \frac{0.03\,(z+0.75)}{z^2 - 1.5\,z + 0.5}, \quad A(z) = 1 - 1.5z^{-1} + 0.5z^{-2},\ B(z) = 0.03 + 0.0225$$

Желаемые полюса задаются по ζ, ω_d:

$$\omega_n = \frac{\omega_d}{\sqrt{1-\zeta^2}},\ r = e^{-\zeta\omega_n T},\ \varphi = \omega_d T,\ p_{1,2} = re^{\pm j\varphi}.$$

Чтобы обеспечить нулевую ошибку на ступень и конечную на линейный вход (тип 1), в знаменатель замкнутой системы включаем интегратор:

$$A_d(z) = (1 - z^{-1})\,(z^2 + a_{1d}z + a_{2d}) \quad \text{с корнями } p_{1,2}.$$

RST-синтез формулируется через диофантово уравнение

$$A(z)\,S(z) + B(z)\,R(z) = A_d(z).$$

Выбираем минимальные порядки, достаточные для точного совпадения степеней:

$$S(z) = s_0 + s_1 z^{-1} \; (s_0 = 1), \quad R(z) = r_0 + r_1 z^{-1} + r_2 z^{-2}.$$

Предфильтр $T(z) = t_0 + t_1 z^{-1} + t_2 z^{-2}$ подбирается из требований

$$H(z) = \frac{B(z)\,T(z)}{A_d(z)}, \quad H(1) = 1, \quad K_v = 0.14.$$

В результате решения $A(z)S(z) + B(z)R(z) = A_d(z)$ получены коэффициенты:

$$S(z) = \boxed{1 + 0.3199\,z^{-1}}, \quad R(z) = \boxed{0 + 7.6094\,z^{-1} - 7.6094\,z^{-2}}.$$

Подбор предфильтра (при $H(1) = 1$ и численной настройке по линейному входу) дал

$$T(z) = \boxed{67.6263 - 115.1116\,z^{-1} + 47.4853\,z^{-2}}.$$

Проверки требований:

$$H(1) = 1 \;\Rightarrow\; \text{ошибка на ступень нулевая}, \qquad K_v \approx 0.1406 \approx 0.14.$$

Сводная таблица коэффициентов:

$S(z)$	$1 + 0.3199\,z^{-1}$
$R(z)$	$0 + 7.6094\,z^{-1} - 7.6094\,z^{-2}$
$T(z)$	$67.6263 - 115.1116\,z^{-1} + 47.4853\,z^{-2}$

Таблица 1 — Итоговые коэффициенты RST

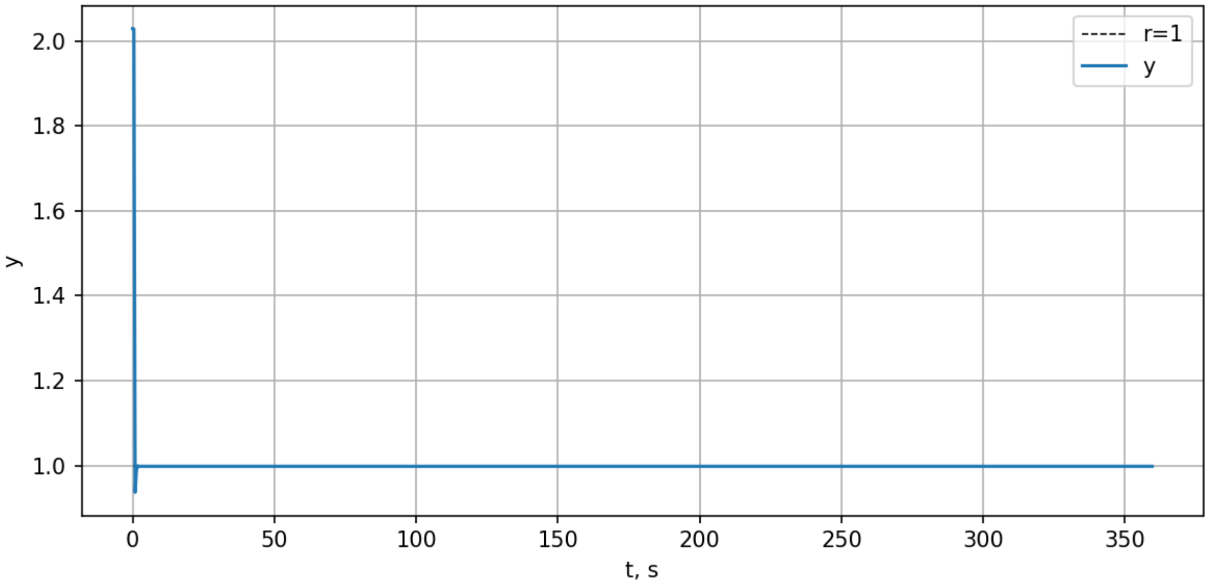

Рисунок 3 — Переходная характеристика для $HG(z)$: полюса по $\zeta = 0.78$, $\omega_d = 4$, нулевая ошибка на ступень, $K_v \approx 0.14$

Ключевой код расчёта и моделирования:

Листинг 1.3 — RST-синтез для $HG(z)$, python/task3.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from pathlib import Path

# HG(z) = 0.03 (z + 0.75) / (z^2 - 1.5 z + 0.5)
B = np.array([0.03, 0.03*0.75])   # b0 + b1 z^{-1}
A = np.array([1.0, -1.5, 0.5])    # 1 + a1 z^{-1} + a2 z^{-2}

#      8
zeta = 0.78
wd = 4.0
T = 0.45
Kv_target = 0.14

#           () +           1
wn = wd / np.sqrt(1 - zeta**2)
r = np.exp(-zeta * wn * T)
phi = wd * T
p1 = r * np.exp(1j*phi)
p2 = r * np.exp(-1j*phi)
A_pair = np.poly([p1, p2]).real  # [1, a1p, a2p]
```

```python
A_int = np.array([1.0, -1.0])     # (1 - z^{-1})
A_d = np.convolve(A_int, A_pair) #      3

#
a0, a1, a2 = A
b0, b1 = B

#     R,S: S(z)=s0 + s1 z^{-1}, R(z)=r0 + r1 z^{-1} + r2 z^{-2}
#       s0=1
s0 = 1.0
#                    z^0..z^-3: A*S + B*R = A_d
#        x = [s1, r0, r1, r2]
M = np.array([
    [0.0, b0,   0.0,  0.0],                    # d0 = a0*s0 + b0*r0
    [a0,  b1,   b0,   0.0],                    # d1 = a1*s0 + a0*s1
       + b1*r0 + b0*r1
    [a1,  0.0,  b1,   b0 ],                    # d2 = a2*s0 + a1*s1
       + b1*r1 + b0*r2
    [a2,  0.0,  0.0,  b1 ],                    # d3 = a2*s1 + b1*r2
], dtype=float)
Y = np.array([
    A_d[0] - a0*s0,
    A_d[1] - a1*s0,
    A_d[2] - a2*s0,
    A_d[3],
], dtype=float)

s1, r0, r1, r2 = np.linalg.solve(M, Y)

#
A_poly = np.poly1d(A)
B_poly = np.poly1d(B)
S_poly = np.poly1d([s0, s1])
R_poly = np.poly1d([r0, r1, r2])
left = (A_poly * S_poly + B_poly * R_poly).c
assert np.allclose(left, A_d, atol=1e-6), "               "

#     T(z)=t0+t1 z^{-1}+t2 z^{-2}: H(1)=1   Kv=0.14
A_d_poly = np.poly1d(A_d)
T1_req = A_d_poly(1.0) / B_poly(1.0)  # t0 + t1 + t2 = T1_req
```

```
def evaluate_t01(t0: float, t1: float):
    t2 = float(T1_req - t0 - t1)
    num = np.polymul(B, np.array([t0, t1, t2]))
    den = A_d
    N = 5000
    k = np.arange(N)
    ramp = k.astype(float)
    y = signal.lfilter(num/den[0], den/den[0], ramp)
    e = ramp - y
    e_ss = np.mean(e[-600:])
    Kv_est = np.inf if e_ss <= 0 else 1.0 / e_ss
    return abs(Kv_est - Kv_target), Kv_est, (t0, t1, t2)

#
best = (np.inf, None, None)
for t0 in np.linspace(-100.0, 100.0, 801):
    for t1 in np.linspace(-100.0, 100.0, 801):
        err, Kv_est, coeffs = evaluate_t01(t0, t1)
        if np.isfinite(Kv_est) and err < best[0]:
            best = (err, Kv_est, coeffs)

err, Kv_est, (t0_best, t1_best, t2_best) = best
span = 10.0
for _ in range(6):
    improved = False
    grid0 = np.linspace(t0_best - span, t0_best + span, 161)
    grid1 = np.linspace(t1_best - span, t1_best + span, 161)
    for t0 in grid0:
        for t1 in grid1:
            e2, Kv2, coeffs2 = evaluate_t01(t0, t1)
            if np.isfinite(Kv2) and e2 < err:
                err, Kv_est = e2, Kv2
                t0_best, t1_best, t2_best = coeffs2
                improved = True
    if not improved:
        break
    span *= 0.35

print(f"RST: S=[{s0:.4f}, {s1:.4f}], R=[{r0:.4f}, {r1:.4f}, {r2
   :.4f}], T=[{t0_best:.5f}, {t1_best:.5f}, {t2_best:.5f}], Kv{
   Kv_est:.5f}, Δ||={err:.3e}")
```

```
#
num_step = np.polymul(B, np.array([t0_best, t1_best, t2_best]))
den = A_d
N = 800
step = np.ones(N)
y_step = signal.lfilter(num_step/den[0], den/den[0], step)

THIS = Path(__file__).resolve()
IMG = THIS.parent.parent / 'images' / 'task3'
IMG.mkdir(parents=True, exist_ok=True)

t = np.arange(N) * T
plt.figure(figsize=(8,4))
plt.plot(t, np.ones_like(t), 'k--', lw=0.8, label='r=1')
plt.plot(t, y_step, label='y')
plt.xlabel('t, s'); plt.ylabel('y'); plt.grid(True); plt.legend()
   ;
plt.tight_layout()
plt.savefig(IMG / 'closed_hg.png', dpi=150)
plt.close()
print('Saved:', IMG / 'closed_hg.png')
```

1.4 Выводы

Получены регуляторы для всех трёх задач. Переходные соответствуют требованиям: в задании 1 — апериодический характер; в задании 2 — поведение, соответствующее целевой модели Даллина; в задании 3 — полюса по ζ,ω_d, нулевая ошибка на ступень и заданная скорость слежения $K_v \approx 0.14$. Формулы синтеза и используемые коэффициенты представлены в тексте и листингах Python.