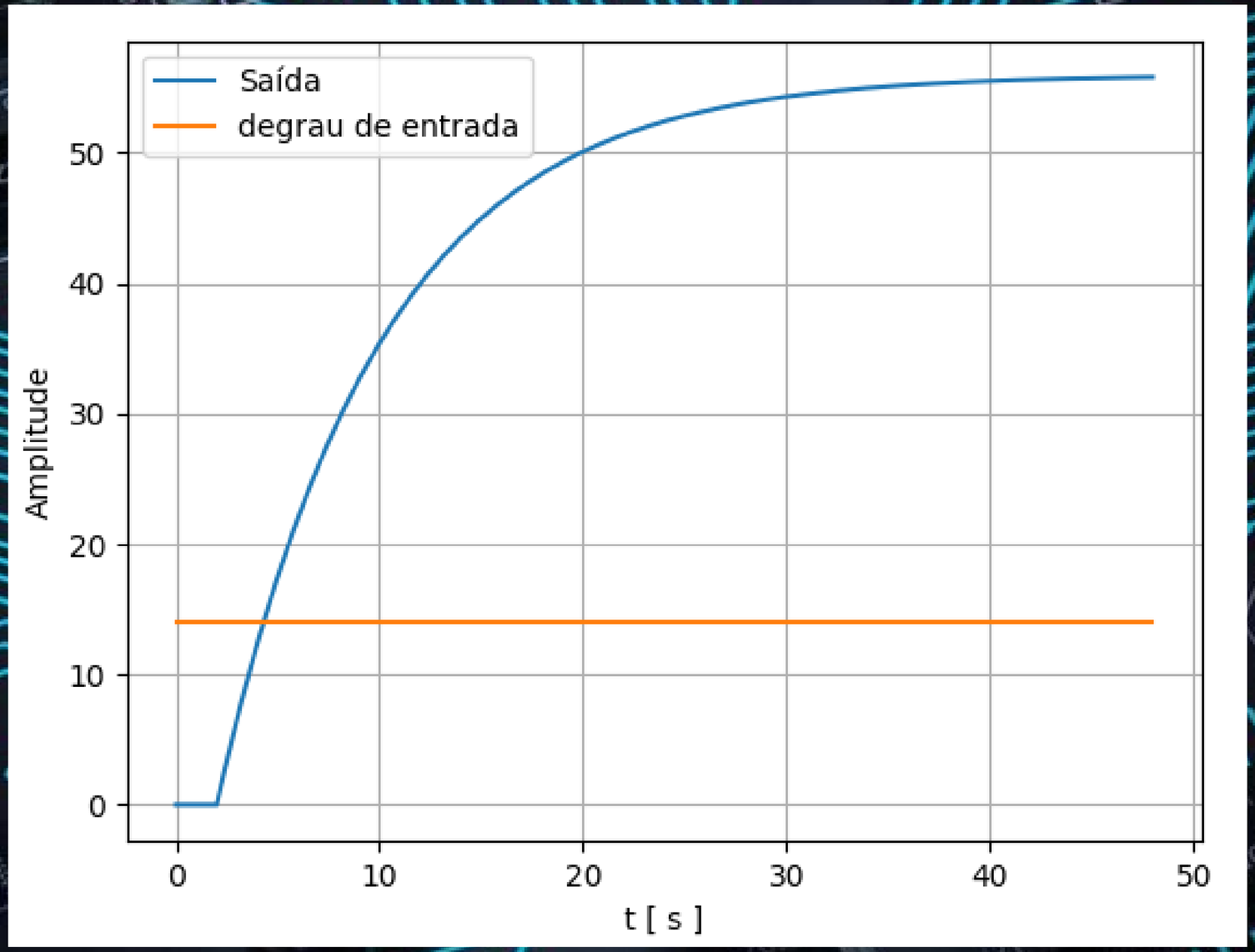
Trabalho de Sistemas Embarcados

GRUPO 14 - INTEGRANTES :

Douglas Brandão de Souza - GEC - 1730
Gabriel Magalhães Reis - GEC - 1737
Pedro Henrique de Souza - GEC - 1721

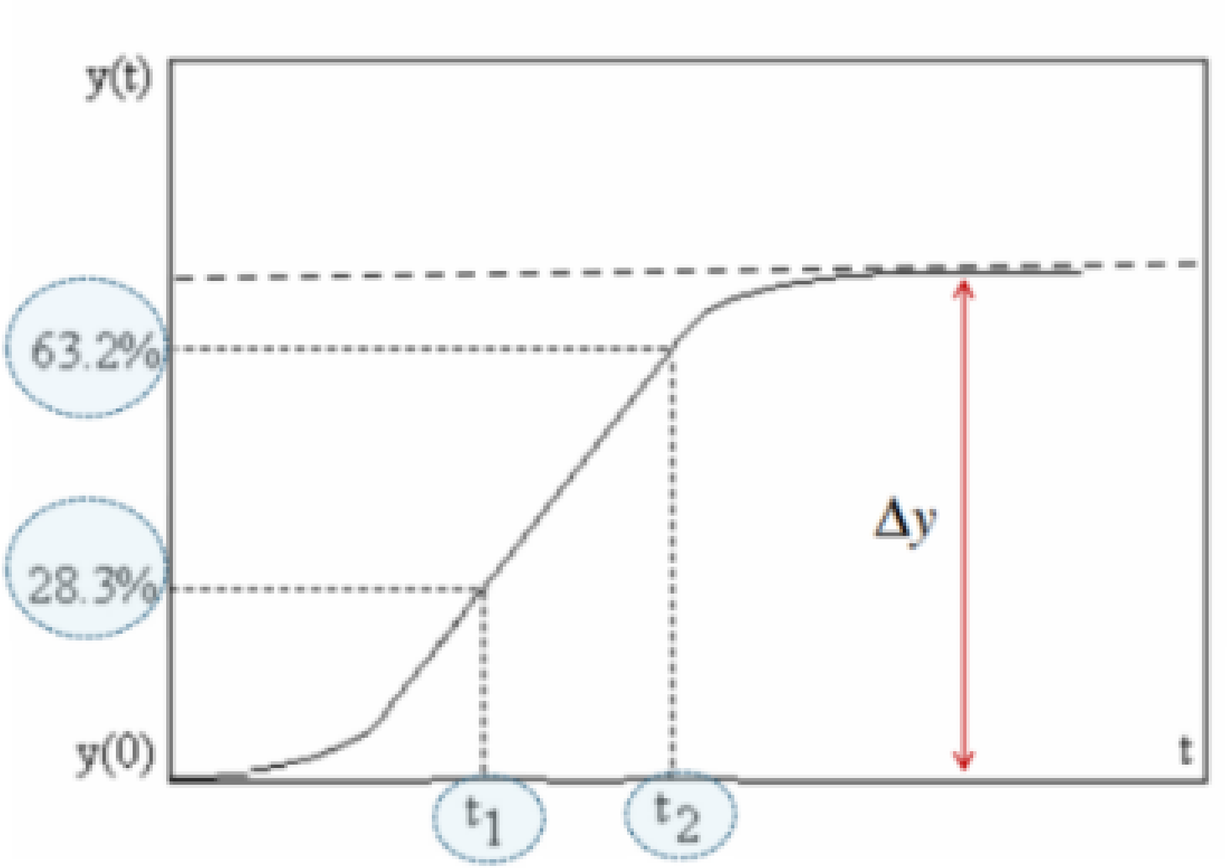
Função de Transferência da Planta:



Valores de K, Tau e Theta:

```
Degrau =  [14]
saida =  [55.82176428]
k =  3.987268876813961
tau =  7.949999999999999
theta =  2.050000000000000007
```

Método de Smith

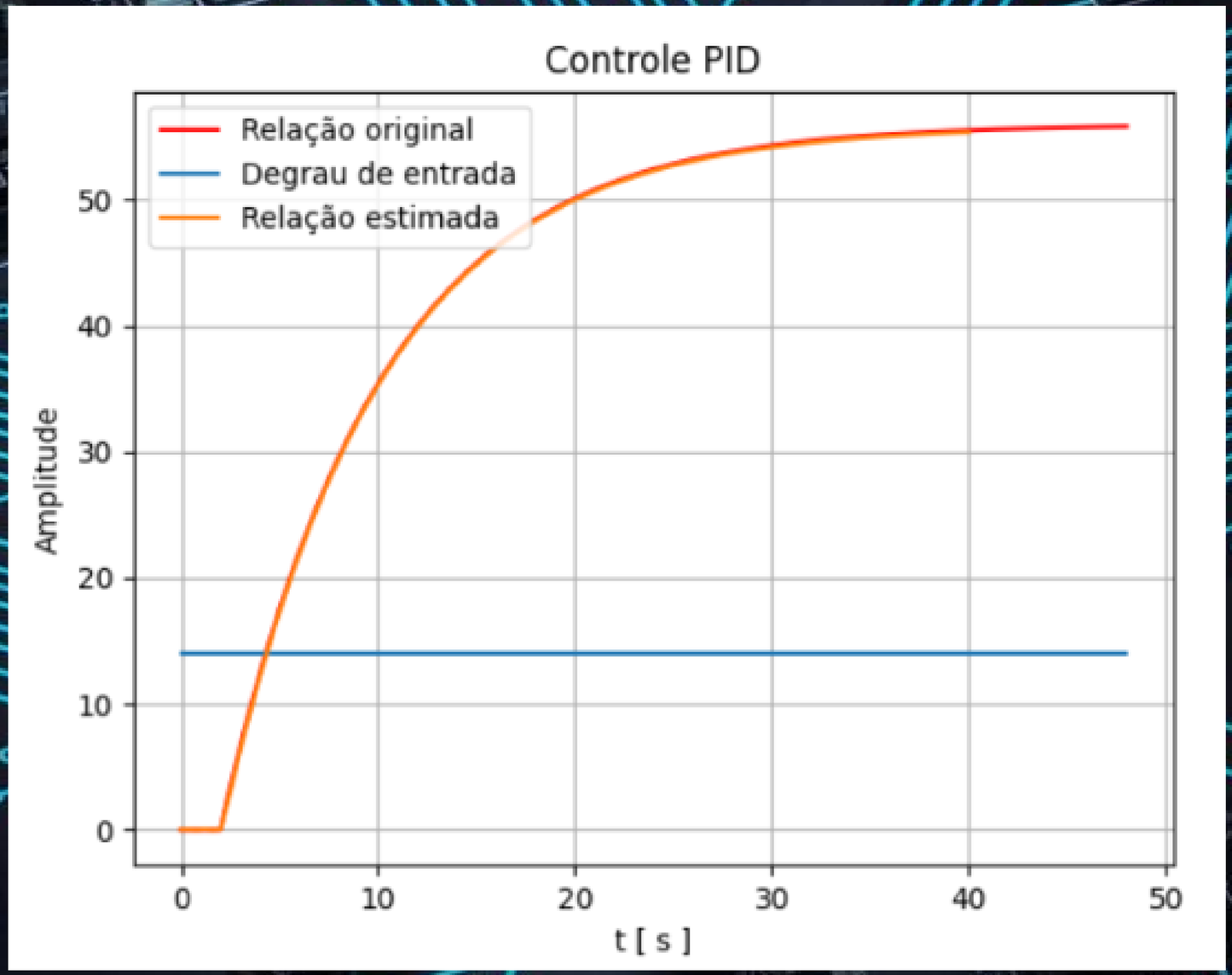


$$K = \frac{\Delta y}{\Delta u}$$

$$\tau = 1.5(t_2 - t_1)$$

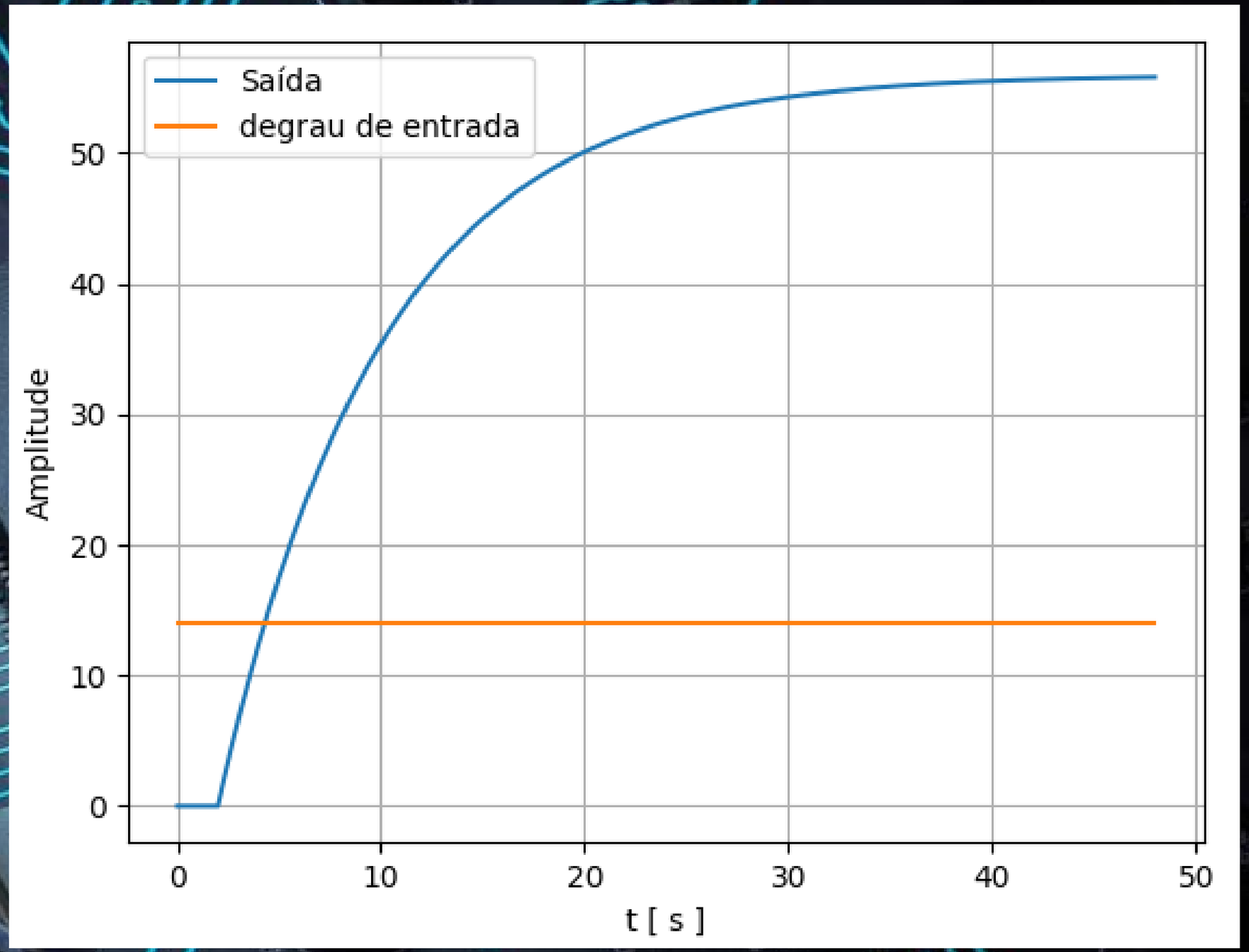
$$\theta = t_2 - \tau$$

Relação Estimada:



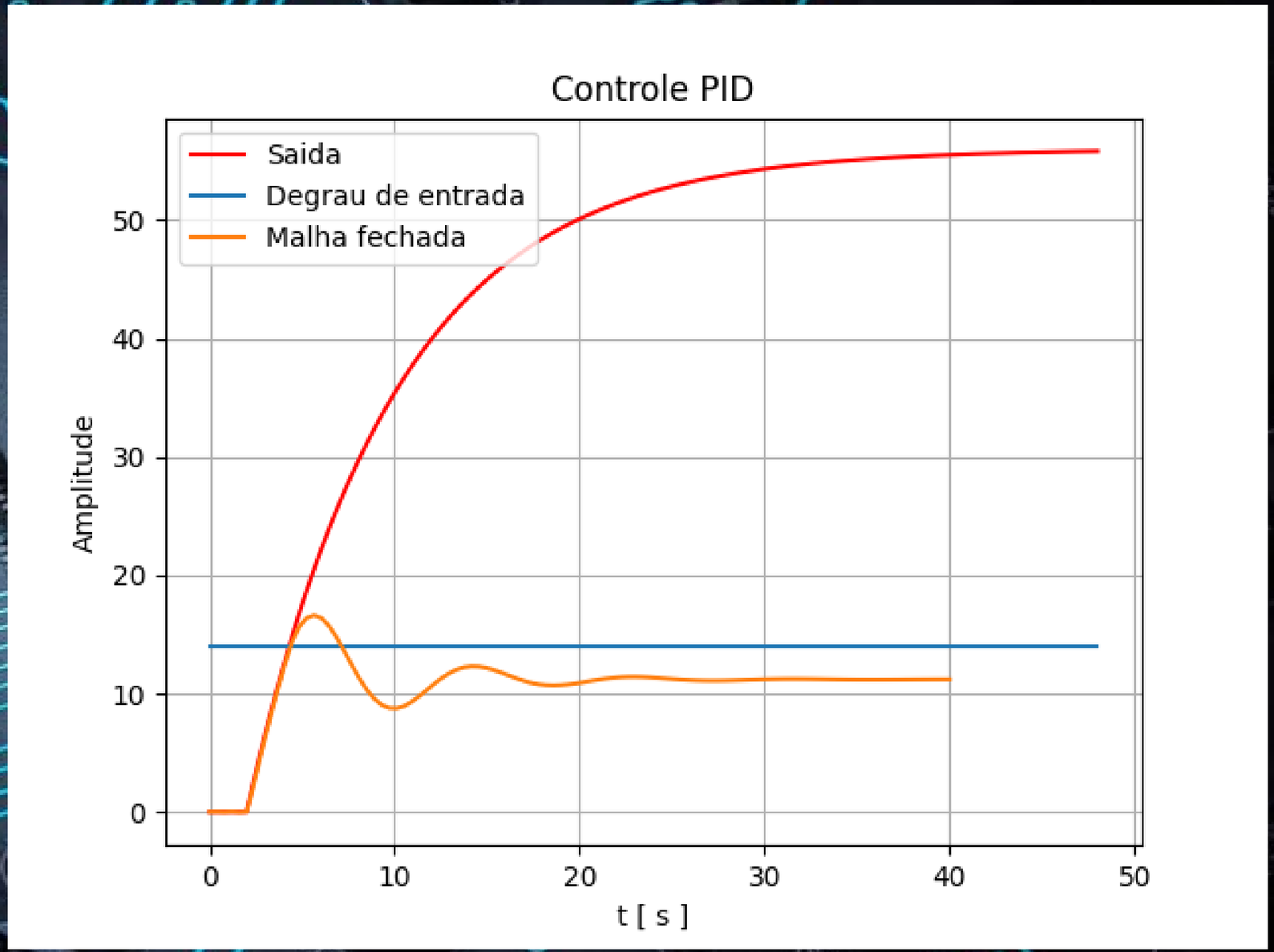
Erro Malha Aberta:

- 55.35 - 14 = 41.35

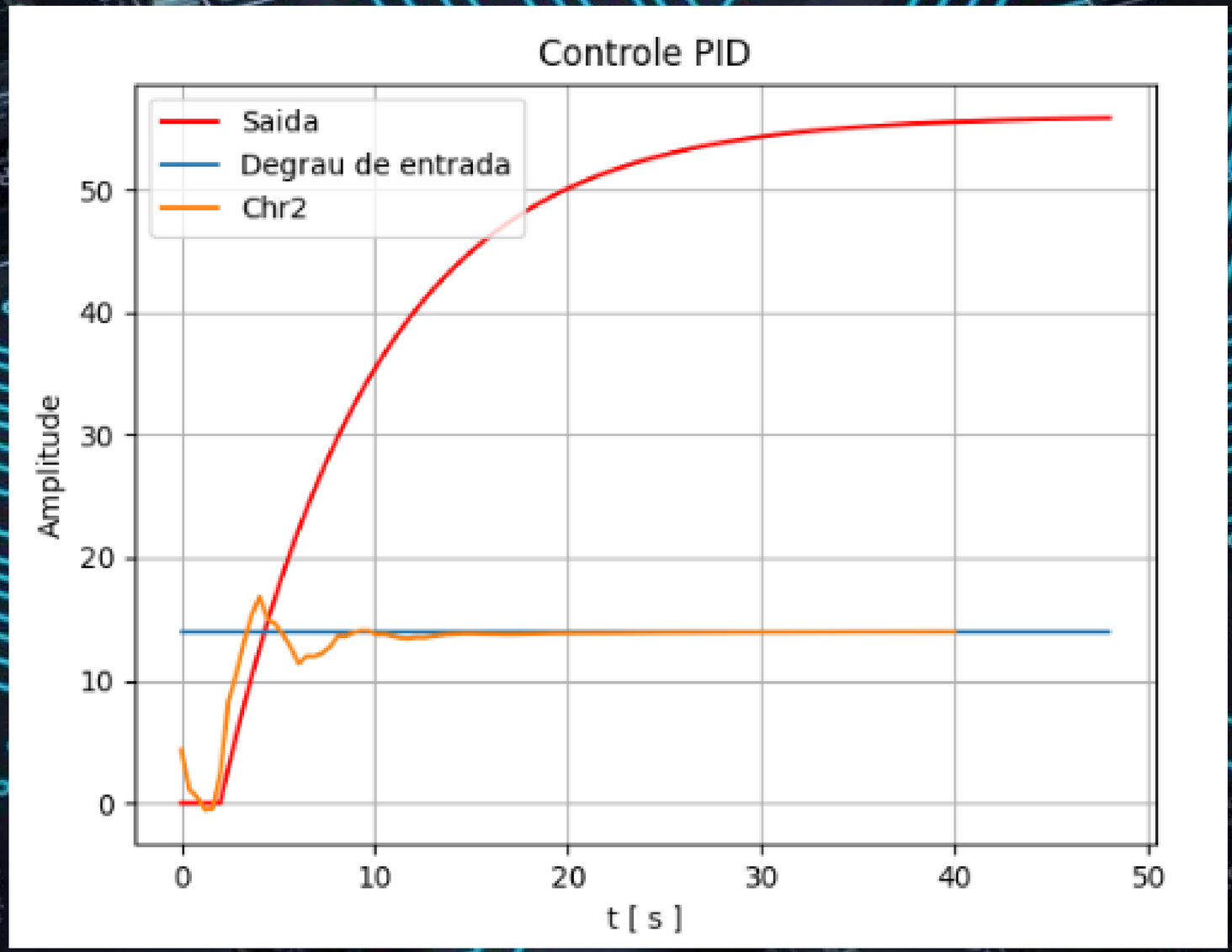


Erro Malha Fechada:

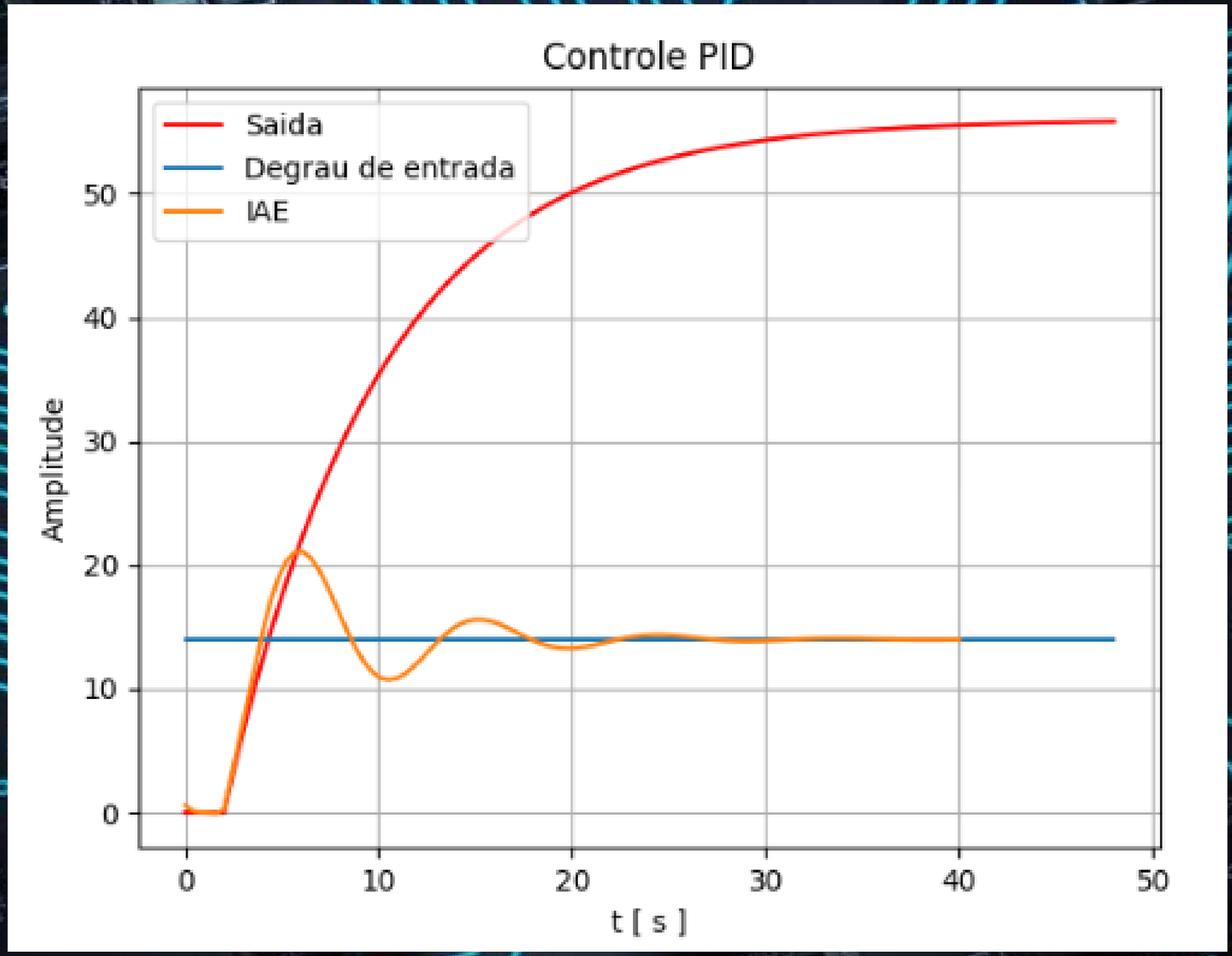
- 14 - 11.2 = 2.8



Método Clássico CHR 2:



Método Novo Integral do Erro:



Ajuste Fino

CHR2 calculado

kp = 0.9239774029002131, Ti = 10.788149999999998, Td = 0.9696500000000002

CHR2 com ajuste

kp = 0.7, Ti = 9, Td = 0.9696500000000002

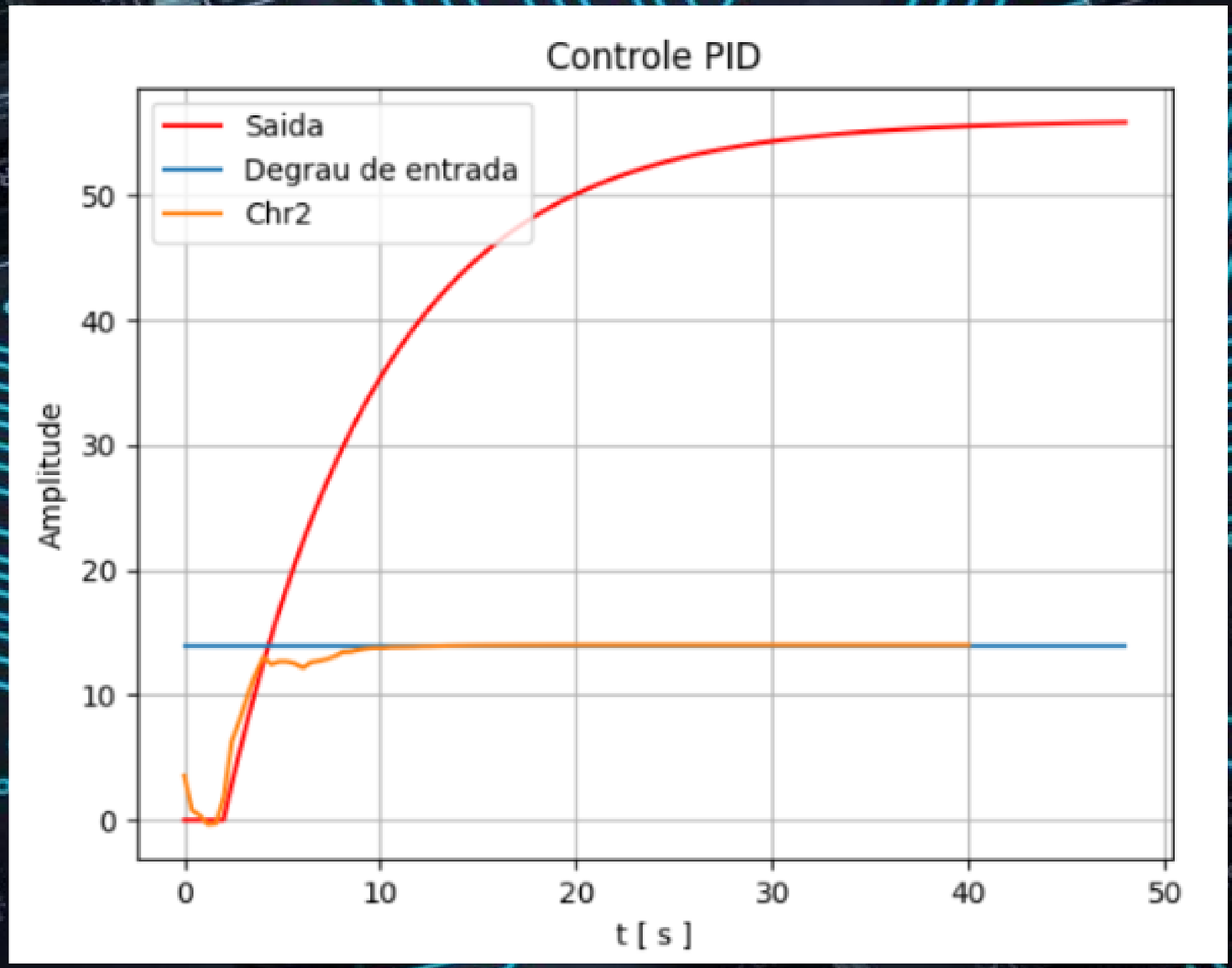
IAE calculado

kp = 1.0227674396882862, Ti = 7.750465376023827, Td = 0.08833333333333332

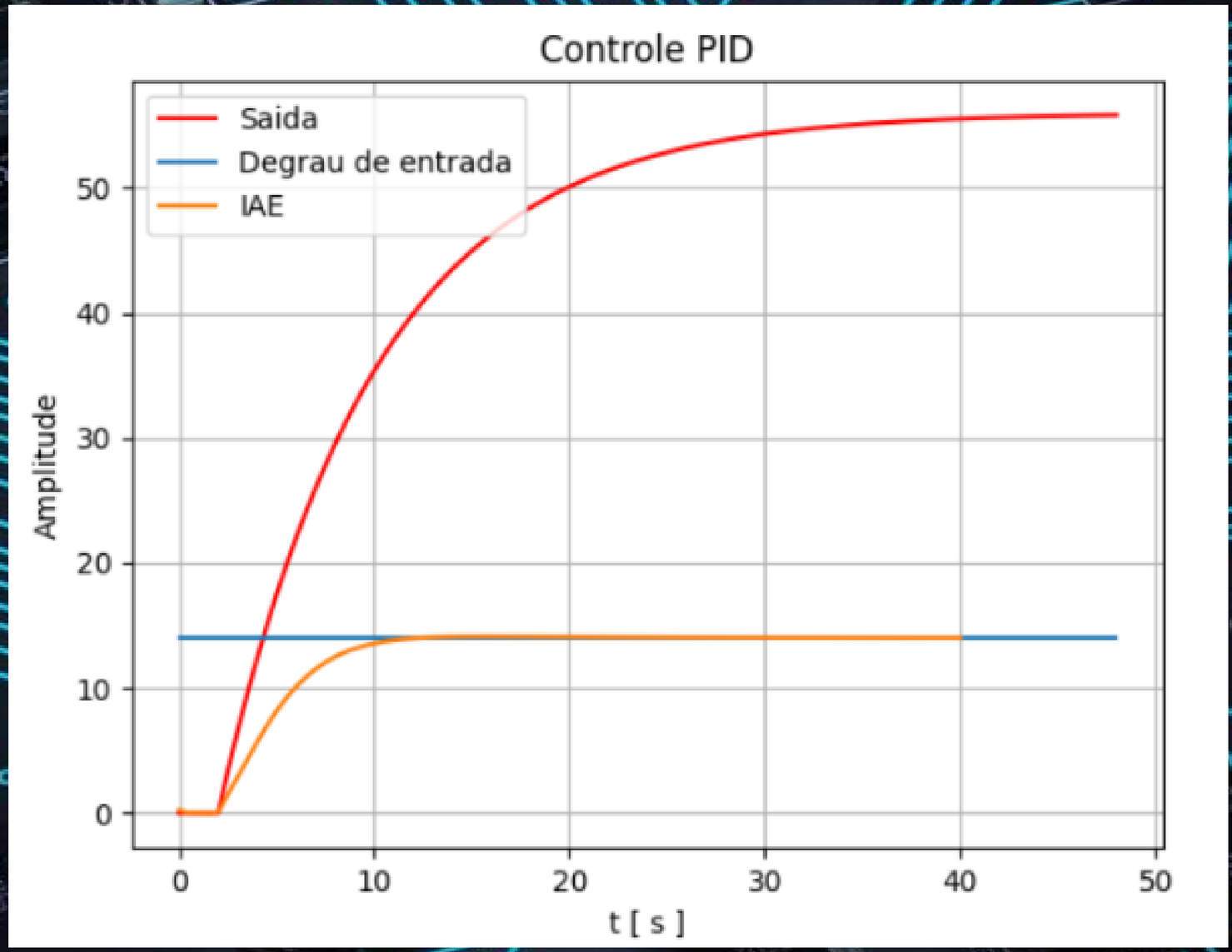
IAE com ajuste

kp = 0.4, Ti = 7.750465376023827, Td = 0.08833333333333332

Ajuste Fino CHR 2:



Ajuste Fino Integral do Erro:



Resultados do Ajuste Fino realizado no Sistema:

Com o ajuste fino conseguimos remover o sobressinal e o sistema entrou em regime permanente mais rápido.

Vantagens do Método Novo em relação ao Método Tradicional:

Com os métodos novos o sistema obteve maior precisão (erro final igual a zero), sem sobre sinal e rápida conversão.

Código - control_pid:

control_pid.py

controle-classico > control_pid.py > ControlPID > calculate_tau

```python
from scipy.io import loadmat
import numpy as np
import control as cnt
import matplotlib.pyplot as plt

class ControlPID:
    def __init__(self) -> None:
        self.mat = None
        self.step = None
        self.output = None
        self.time = None
        self.t1 = None
        self.t2 = None

    def loadmat(self, transfer_function):
        self.mat = loadmat(transfer_function)

    def set_step(self):
        self.step = self.mat.get('degrau')
        print("Degrau = ", self.step[0])

    def set_output(self):
        self.output = self.mat.get('saida')
        print("saida = ", self.output[-1])

```

```
21
22      def set_output(self):
23          self.output = self.mat.get('saida')
24          print("saida = ", self.output[-1])
25
26      def set_time(self):
27          self.time = self.mat.get('t')
28
29      def calculate_k(self):
30          delta_y = self.output[-1]
31          delta_u = self.step[0]
32          return float(delta_y / delta_u)
33
34      def calculate_tau(self):
35          #print('Saida = ', self.output[-1])
36
37          y_t1 = 0.283*self.output[-1]
38          y_t2 = 0.632*self.output[-1]
39
40          #print('y(t1) 0.283*saida = ', y_t1)
41          #print('y(t2) 0.632*saida = ', y_t2)
```

```
42
43          closest_from_y_t1 = self.closest(self.output, y_t1)
44          closest_from_y_t2 = self.closest(self.output, y_t2)
45
46          self.t1 = self.time[0][int(np.where(self.output == closest_from_y_t1)[0])]
47          self.t2 = self.time[0][int(np.where(self.output == closest_from_y_t2)[0])]
48
49          #print('t1 = ', self.t1)
50          #print('t2 = ', self.t2)
51
52          return float(1.5 * (self.t2 - self.t1))
53
54      def calculate_theta(self, tau):
55          return float(self.t2 - tau)
56
57      def closest(self, lst, K):
58          lst = np.asarray(lst)
59          idx = (np.abs(lst - K)).argmin()
60          return lst[idx]
61
62      def transfer_function(self, k, tau, theta):
63          num = np.array([k])
64          den = np.array([tau, 1])
65          H = cnt.tf(num, den)
66          n_pade = 20
67          num_pade, den_pade = cnt.pade(theta, n_pade)
68          H_pade = cnt.tf(num_pade, den_pade)
69          return cnt.series(H , H_pade)
```

```
70
71      def feedback(self, Hs):
72          return cnt.feedback(Hs, 1)
73
74      def CHR2(self, k, tau, theta):
75          kp = 0.95*tau/(k*theta)
76          ti = 1.357*tau
77          td = 0.473*theta
78
79          return kp, ti, td
80
81      def integral_erro(self, k, tau, theta):
82          kp = (1 / (theta/tau) + 0.2) / k
83          ti = ((0.3 * (theta / tau) + 1.2) / ((theta / tau) + 0.08)) * theta
84          td = (1 / (90 * (theta / tau))) * theta
85
86          return kp, ti, td
87
88      def controlador_pid(self, kp, Ti, Td, Hs):
89          # Controlador proporcional
90          numkp = np. array ([kp])
91          denkp = np. array ([1])
92          #integral
93          numki = np. array ([kp])
94          denki = np. array ([Ti,0])
95          #derivativo
96          numkd = np. array ([kp*Td,0])
97          denkd = np. array ([1])
```

```
99          #Construindo o controlador PID
100         Hkp = cnt.tf(numkp , denkp)
101         Hki=cnt.tf(numki , denki)
102         Hkd=cnt.tf(numkd , denkd)
103         Hctrl1 = cnt.parallel (Hkp , Hki)
104         Hctrl = cnt.parallel (Hctrl1 , Hkd)
105         Hdel = cnt.series (Hs , Hctrl)
106         #Fazendo a realimentação
107         Hcl = cnt.feedback(Hdel, 1)
108         return Hcl
109
110     def plot_output(self, Hs):
111         t = np.linspace(0, 40, 100)
112         t, y = cnt.step_response(Hs * self.step[0], t)
113
114         plt.plot(self.time.T, self.output, color='r', label='Saída')
115         plt.plot(self.time.T, self.step, label='Degrau de entrada')
116         plt.plot(t, y)
117
118         plt.xlabel (' t [ s ] ')
119         plt.ylabel('Amplitude')
120         plt.title("Controle PID")
121         plt.legend(loc="upper left")
122
123         plt.grid()
124         plt.show()
```

Código - main:

main.py M

controle-classico > main.py > main

```python
from control_pid import ControlPID

def main():
    control_pid = ControlPID()

    control_pid.loadmat('TransferFunction14.mat')
    control_pid.set_step()
    control_pid.set_output()
    control_pid.set_time()

    k = control_pid.calculate_k()
    print('k = ', k)

    tau = control_pid.calculate_tau()
    print('tau = ', tau)

    theta = control_pid.calculate_theta(tau)
    print('theta = ', theta)

    Hs = control_pid.transfer_function(k, tau, theta)

```

```
21
22      #Fazendo a realimentação
23      # Hcl = control_pid.feedback(Hs)
24      Hcl = Hs
25
26      erro_malha_aberta = 55.35 - 14
27      erro_malha_fechada = 14 - 11.2
28
29      control_pid.plot_output(Hcl)
30
31
32      kp, ti, td = control_pid.CHR2(k, tau, theta)
33      print(kp, ti, td)
34
35      for i in range(5, 15):
36          kp = i*0.1
37          #kp = 0.6 ou 0.7 é boa
38          Chr2 = control_pid.controlador_pid(kp, ti, td, Hs)
39          #control_pid.plot_output(Chr2)
40
```

```
40
41      Chr2 = control_pid.controlador_pid(0.7, ti, td, Hs)
42      control_pid.plot_output(Chr2)
43      kp, ti, td = control_pid.integral_erro(k, tau, theta)
44      print(kp, ti, td)
45
46      for i in range(2, 15):
47          kp = i*0.1
48          #kp = 0.6 ou 0.7 é boa
49
50      integral_erro = control_pid.controlador_pid(0.4, ti, td, Hs)
51
52      control_pid.plot_output(integral_erro)
53
54
55  if __name__=="__main__":
56      main()
```


FIM
A Magia dos Sistemas Embarcados
Obrigado!
@codecrush.com.br