UNIVERSIDADE FEDERAL DO TRIÂNGULO MINEIRO

Instituto de Ciências Tecnológicas e Exatas

Lucas Felipe Dutra

Utilização do Critério das Áreas e Método Passo-a-Passo

Dinâmica de Sistemas Elétricos

Professor: Fabrício Augusto Matheus Moura

Disciplina: Dinâmica de Sistemas Elétricos

Uberaba-MG

08/05/2019

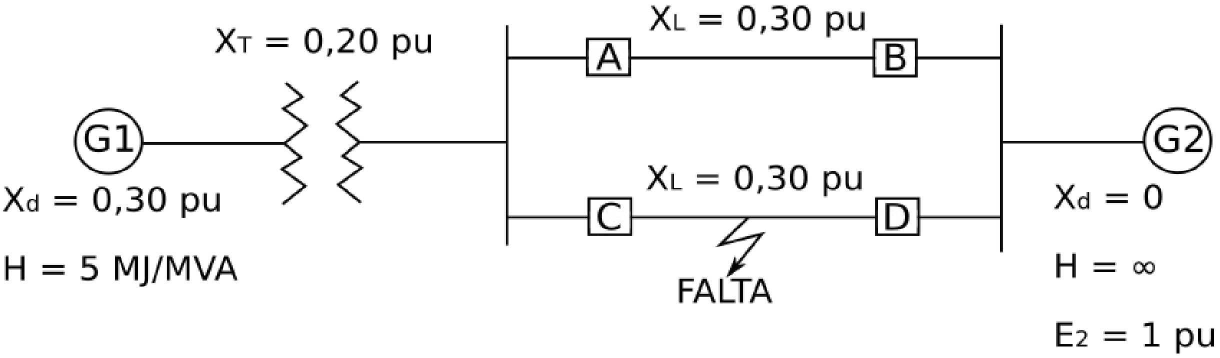
Sumário

1. ANÁLISES

O presente trabalho visa observar o comportamento do circuito apresentado na figura 1 no que se refere ao comportamento do ângulo de chaveamento crítico com relação a variação da potência mecânica, a variação do ângulo delta com relação ao tempo para diferentes potências mecânicas, o comportamento do tempo de chaveamento crítico com relação a constante de inércia.

Para efetuar todos os calculos e plotar os gráficos que virão a seguir, foi feito um código utilizando a linguagem Python. Todo esse código será colocado na proxima seção, porém ele também se encontra disponível em meu github no link: https://github.com/LucasFDutra/Dinamica

Figura 1: Sistema a ser análisado



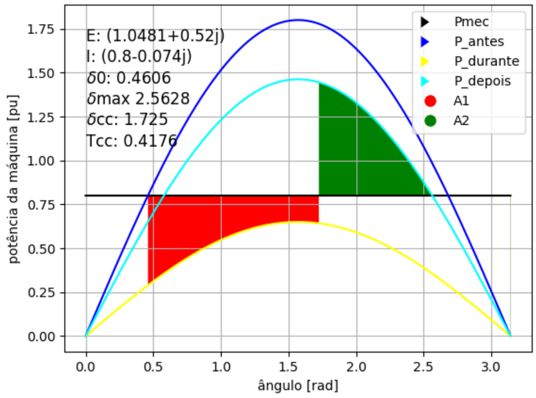
Fonte: do autor, 2019

Para o sistema em questão foram aplicados cinco valores de potência mecânicas diferentes e para cada um desses valores foi aplicado o método das áreas iguais, assim obtendo o ângulo de chaveamento crítico e também o tempo de chaveamento crítico, o qual foi calculado utilizando o método passo a passo (todo o código referente ao método passo a passo pode ser visto no arquivo tabela.py) e assim, foram obtidos os gráficos das figuras 2 à 6, que contam com informações como:

- Força eletromotriz da máquina;
- Corrente no sistema;
- Ângulo inicial do sistema;
- Ângulo de chaveamento crítico;
- Ângulo máximo;
- Curvas de potência eletrica x delta de cada instante do circuito;
- Áreas antes (vermelho) e após (verde) o chaveamento.

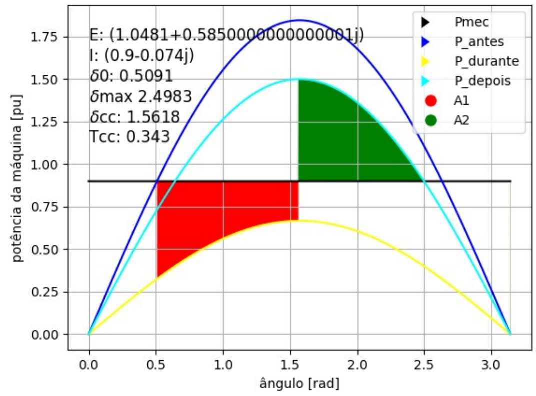
Também foram montados, através de interpolação, os gráficos referentes à variação do ângulo delta com relação ao tempo para cada uma das cinco potências mecânicas. Tais gráficos se encontram na figura 7 e a curva em vermelho é referente à potência mais baixa e o a curva em rosa é referente à potência mais alta.

Figura 3: Potência mecânica igual a 0.8



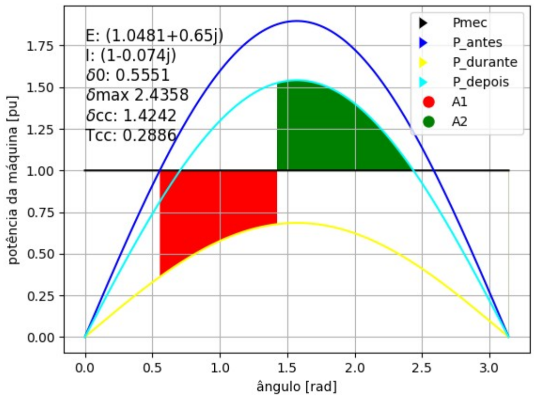
Fonte: do autor, 2019

Figura 2: Potência mecânica igual a 0.9



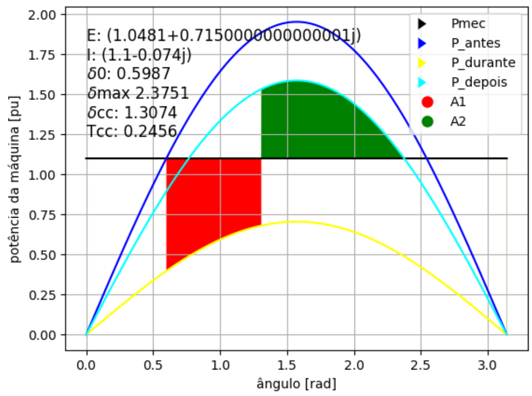
Fonte: do autor, 2019

Figura 4: Potência mecânica igual a 1



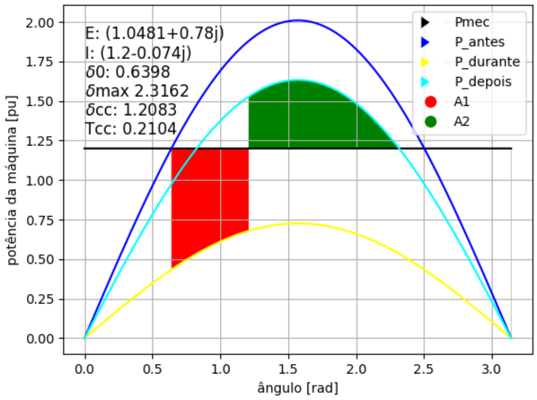
Fonte: do autor, 2019

Figura 5: Potência mecânica igual a 1.1



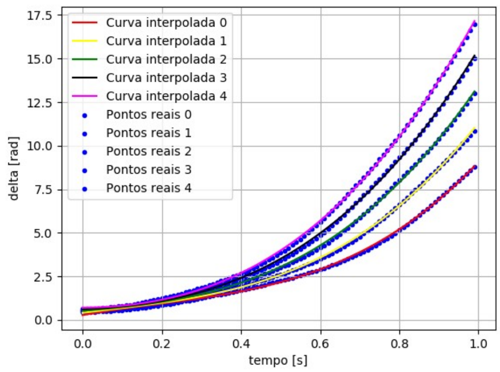
Fonte: do autor, 2019

Figura 6: Potência mecânica igual a 1.2



Fonte: do autor, 2019

Figura 7: Variação do ângulo delta em relação ao tempo



Fonte: do autor, 2019

Observando as figuras acima e os resultados mostrados nelas, podemos constatar que existem alterações em todos os parâmetros do circuito, porém as duas análises mais interessante ao variarmos o valor da potência mecânica é no ângulo de chaveamento crítico e no tempo de chaveamento crítico.

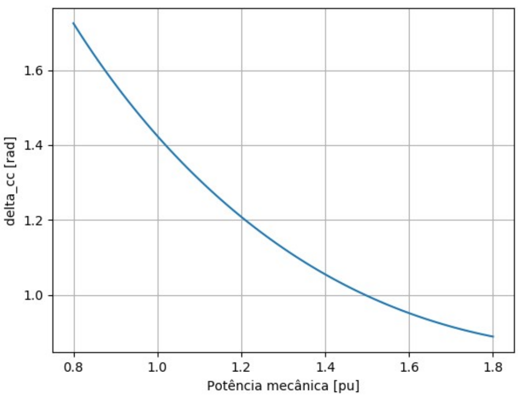
Como pode ser observado nas figuras 2 à 6 o ângulo de chaveamento crítico diminui conforme aumentamos o valor da potência mecânica. O motivo para isso pode ser descrito tanto pela equação que obtemos para o cálculo de utilizando a teoria de áreas iguais, quanto pela análise visual dos gráficos em conjunto com a teoria das áreas iguais.

Como temos os gráficos em mãos, vamos fazer a análise pelo método gráfico em conjunto com a teoria das áreas iguais. A teoria das áreas iguais diz que a área formada entre as curvas de potência mecânica e a potência elétrica durante a falta (área em vermelho) deve ser igual à área formada entre as curvas de potência mecânica e a potência elétrica pós falta (área em verde). Logo se olharmos para o gráfico vemos que a disponibilidade de espaço para a área verde diminui conforme aumentamos o valor da potência mecânica, logo a área em vermelho também precisa diminuir, por tanto para manter o equilíbrio deve-se diminuir a área vermelha no sentido horizontal, o que ocasiona uma diminuição do ângulo de chaveamento crítico.

Já o tempo de chaveamento crítico não sofre grandes alterações devido à potência mecânica, como podemos observar na figura 7 as curvas entre os valores de 0 a 3 para δ_{cc} não apresentam muita diferenças.

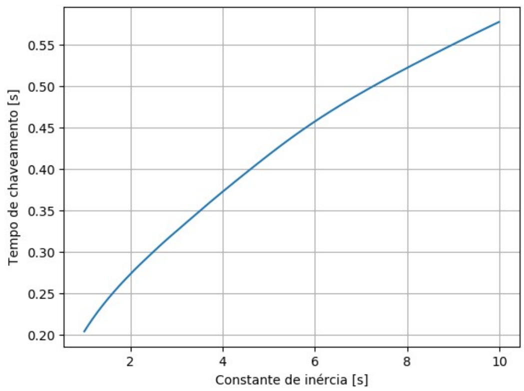
Para completar o estudo, foram feitos os gráficos das figuras 8 e 9. Sendo que o gráfico da figura 8 contempla a análise feita anteriormente de que o ângulo de chaveamento crítico decresce com o aumento da potência mecânica. E o gráfico da figura 9 representa a variação do tempo de chaveamento crítico com relação à constante de inércia.

Figura 8: Ângulo de chaveamento critico em relação à potência mecânica



Fonte: do autor, 2019

Figura 9: Tempo de chaveamento crítico em relação a constante de inércia



Fonte: do autor, 2019

E agora com o gráfico da figura 9 podemos ver que o fator dominante para a variação do tempo de chaveamento crítico é a constante de inércia. Sendo que a variação do Tcc é diretamente proporcional à variação da constante de inércia. E como visto em sala de aula, é conveniente que o tempo de chaveamento crítico seja mais elevado, e para atingir tal objetivo é necessário aumentar a constante de inércia da máquina, o que é feito aumentando a massa da mesma através dos volantes inerciais.

2. CÓDIGOS

Arquivo main.py

```
import tabela as tb
import calculos as ca
import parametros as pr
import graficos as gf

X = [complex(0,0.65), complex(0,1.8), complex(0,0.8)]
U = 1
H = 5
F = 60

cont = 0
delta_list = []
f_list = []
# plotando gráficos para 5 S diferentes
S_list = [0.8, 0.9, 1, 1.1, 1.2]
for i in S_list:
    S = complex(i, 0.074)
    I, E, Pmec, Pmax_list, DELTA_0, DELTA_cc, DELTA_max = ca.Calculos(X, U, S, H)
    t, deltas = tb.Tabela(H, F, Pmec, Pmax_list, DELTA_0)
    delta_list.append(deltas)
    y = ca.Interpolacao(t,deltas)
    f = (y[3]) + (y[2]*t) + (y[1]*(t**2)) + (y[0]*(t**3))
    f_list.append(f)
    Tcc = ca.SolvePoly(DELTA_cc, y)
    gf.Potencia_x_Delta(E, I, Pmec, Pmax_list, DELTA_0, DELTA_cc, DELTA_max, Tcc, cont)
    cont+=1
S = complex(0.8,0.074)
gf.Delta_x_Tempo(t, delta_list, f_list)
gf.Delta_cc_x_Pmec(X, U, S, H, Pmax_list[0], cont)
gf.Tcc_x_H(X, U, S, F, cont)
```

Arquivo parametros.py

```
def Parametro_X():
    Xr = [float(i) for i in input('Digite as partes REAIS de X: ').split(' ')]
    Xi = [float(i) for i in input('Digite as partes IMG de X: ').split(' ')]
    X = [complex(Xr[i], Xi[i]) for i in range(len(Xr))]
    return X

def Parametro_U():
    Ur = float(input('digite a parte REAL de U: '))
    Ui = float(input('digite a parte IMG de U: '))
    U = complex(Ur, Ui)
    return U

def Parametro_S():
    P = float(input('digite o valor da potência ativa: '))
    Q = float(input('digite o valor da potência reativa: '))
    S = complex(P,Q)
    return S

def Parametro_H():
    H = float(input('digite o valor da constante de inércia da máquina: '))
    return H

def Parametro_F():
    f = float(input('digite o valor da frequência do sistema: '))
    return f
```

Arquivo calculos.py

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
# from scipy.interpolate import interp1d

def Calculos(X, U, S, H):
    Pmec = S.real
    I = S/U
    I = I.conjugate()
    E = U+(X[0]*I)
    DELTA_0 = np.angle(E)
```

```
    Pmax_list = []
    for i in range(3):
        Pmax = (abs(E)*abs(U))/(abs(X[i]))
        Pmax_list.append(Pmax)
    DELTA_max = np.pi-np.arcsin(Pmec/Pmax_list[2])
    DELTA_cc = np.arccos(((Pmec*(DELTA_0-DELTA_max))+(Pmax_list[1]*np.cos(DELTA_0))-
(Pmax_list[2]*np.cos(DELTA_max)))/(Pmax_list[1]-Pmax_list[2]))
    return I, E, Pmec, Pmax_list, DELTA_0, DELTA_cc, DELTA_max

def Interpolacao(t,delta):
    f = np.polyfit(t,delta,deg=3)
    return f

def SolvePoly(y, p):
    p[3]-=y
    x = np.roots(p)
    for i in range(len(x)):
        if x[i].imag == 0 and x[i].real > 0:
            raiz = x[i].real
    return raiz
```

Arquivo tabela.py

```
import numpy as np

def Tabela(H, F, Pmec, Pmax_list, DELTA_0):
    DELTA_t = 0.01
    k = (np.pi*F*(DELTA_t**2))/H

    Pe_01 = Pmax_list[1]*np.sin(DELTA_0)
    Pa_01 = Pmec-Pe_01
    Pa_02 = Pa_01/2
    K_Pa_02 = k*Pa_02
    DELTA_delta = K_Pa_02
    DELTA = DELTA_0

    deltas = [DELTA]

    t = np.arange(0,1,DELTA_t)
    intervalo = len(t)

    for i in range(intervalo-1):
        Pmax = Pmax_list[1]
        sen_d = np.sin(DELTA)
        Pe = Pmax*sen_d
        Pa = Pmec-Pe
        K_Pa = k*Pa
        DELTA_delta = DELTA_delta+K_Pa
        DELTA = DELTA+DELTA_delta
        deltas.append(DELTA)

    return t, deltas
```

Arquivo graficos.py

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.lines import Line2D
import calculos as ca
import tabela as tb

def Potencia_x_Delta(E, I, Pmec, Pmax_list, DELTA_0, DELTA_cc, DELTA_max, Tcc, cont):
    DELTA = np.arange(0,np.pi,0.001)

    p_mec = Pmec+(DELTA-DELTA)
    colors = ['black', 'blue', 'yellow', 'cyan']
    potencias = [p_mec]

    for i in range(4):
        if i>0:
            potencias.append(Pmax_list[i-1]*np.sin(DELTA))
        plt.plot(DELTA, potencias[i], color=colors[i])

    plt.fill_between(DELTA, potencias[0], potencias[2],
```

```
                where = DELTA>=DELTA_0,
                facecolor='red',
                interpolate=True)
plt.fill_between(DELTA, potencias[0], potencias[2],
                where = DELTA>DELTA_cc,
                facecolor='w',
                interpolate=True)

plt.fill_between(DELTA, potencias[0], potencias[3],
                where = DELTA>=DELTA_cc,
                facecolor='green',
                interpolate=True)
plt.fill_between(DELTA, potencias[0], potencias[3],
                where = DELTA>DELTA_max,
                facecolor='w',
                interpolate=True)

font_size = 12
plt.text(0, (Pmax_list[0]-(font_size*1/100)),
        'E: '+ str(E),
        {'color': 'black', 'fontsize': font_size})
plt.text(0, (Pmax_list[0]-(font_size*2/100)),
        'I: '+ str(I),
        {'color': 'black', 'fontsize': font_size})
plt.text(0, (Pmax_list[0]-(font_size*3/100)),
        '$\delta$0: '+ str(round(DELTA_0, 4)),
        {'color': 'black', 'fontsize': font_size})
plt.text(0, (Pmax_list[0]-(font_size*4/100)),
        '$\delta$max '+ str(round(DELTA_max, 4)),
        {'color': 'black', 'fontsize': font_size})
plt.text(0, (Pmax_list[0]-(font_size*5/100)),
        '$\delta$cc: '+ str(round(DELTA_cc, 4)),
        {'color': 'black', 'fontsize': font_size})
plt.text(0, (Pmax_list[0]-(font_size*6/100)),
        'Tcc: '+ str(round(Tcc, 4)),
        {'color': 'black', 'fontsize': font_size})

legend = [
    Line2D([0],[0],
        marker=5,
        color='w',
        label='Pmec',
        markerfacecolor=colors[0],
        markersize=10),
    Line2D([0],[0],
        marker=5,
        color='w',
        label='P_antes',
        markerfacecolor=colors[1],
        markersize=10),
    Line2D([0],[0],
        marker=5,
        color='w',
        label='P_durante',
        markerfacecolor=colors[2],
        markersize=10),
    Line2D([0],[0],
        marker=5,
        color='w',
        label='P_depois',
        markerfacecolor=colors[3],
        markersize=10),
    Line2D([0],[0],
        marker='o',
        color='w',
        label='A1',
        markerfacecolor='r',
        markersize=10),
    Line2D([0],[0],
        marker='o',
        color='w',
        label='A2',
        markerfacecolor='g',
        markersize=10),]
```

```python
    plt.xlabel('ângulo [rad]')
    plt.ylabel('potência da máquina [pu]')
    plt.grid(True)
    plt.legend(handles=legend, loc='best')
    plt.savefig('Potencia_x_Delta_' + str(cont) + '.png')
    plt.close()

def Delta_x_Tempo(t, delta, y):
    dim = len(delta)
    colors = ['red', 'yellow', 'green', 'black', 'magenta']

    for i in range(dim):
        plt.scatter(t, delta[i], color='blue', marker='.', label='Pontos reais '+str(i))
        plt.plot(t, y[i], color=colors[i], label='Curva interpolada '+str(i))
    plt.xlabel('tempo [s]')
    plt.ylabel('delta [rad]')
    plt.grid(True)
    plt.legend(loc='best')
    plt.savefig('Delta_x_Tempo.png')
    plt.close()

def Delta_cc_x_Pmec(X, U, S, H, Pmax, cont):
    pm = S.real
    t = np.arange(pm,Pmax*0.9,0.01)
    Pmec_list = []
    Deltas = []
    for i in t:
        S = complex(i,S.imag)
        _, _, Pmec, _, _, DELTA_cc, _ = ca.Calculos(X, U, S, H)
        Pmec_list.append(Pmec)
        Deltas.append(DELTA_cc)
    plt.plot(Pmec_list, Deltas)
    plt.xlabel('Potência mecânica [pu]')
    plt.ylabel('delta_cc [rad]')
    plt.grid(True)
    plt.savefig('Delta_cc_x_Pmec_' + str(cont) + '.png')
    plt.close()

def Tcc_x_H(X, U, S, F, cont):
    h = np.arange(1,10,0.01)
    Tcc_list = []
    for i in h:
        _, _, Pmec, Pmax_list, DELTA_0, DELTA_cc, _ = ca.Calculos(X, U, S, i)
        t, deltas = tb.Tabela(i, F, Pmec, Pmax_list, DELTA_0)
        y = ca.Interpolacao(t,deltas)
        f = (y[3]) + (y[2]*t) + (y[1]*(t**2)) + (y[0]*(t**3))
        Tcc = ca.SolvePoly(DELTA_cc, y)
        Tcc_list.append(Tcc)
    plt.plot(h, Tcc_list)
    plt.xlabel('Momento de inércia [kg.m^2]')
    plt.ylabel('Tempo de chaveamento [s]')
    plt.grid(True)
    plt.savefig('Tcc_x_H_' + str(cont) + '.png')
    plt.close()
```