Institute for electrical energy systems
Chairman: Univ.-Prof. Dr.-Ing. Matthias Luther

Student Research Paper

Critical clearing time of synchronous generators

Author: B. Eng. Maximilian Köhler
23176975

Supervisor: M. Sc. Ilya Burlakin

Submission date: March 31, 2024

Todo list

Contents

1 Introduction

Bullet points for the thesis from Ilya:

- Swing equation of synchronous generators
- Solving the Swing equation with the help of Python -> Solving of second order ODEs
- Equal-area criterion -> Derivation of the equations
- Simulation of a fault -> applying the equal-area criteria with the help of Python.
- Comparison between analytical and (numerical) simulation results

Introduction via [1] and other standard literature like [2]–[6]. Need for understanding of Transient stability and therefore critical pole angle and fault clearing time assessment: Running and maintaining the electrical grid; Adding virtual inertia in FACTs and HVDC; Better and faster predicting, due to shorter (critical) fault clearing times; .

[MK1]: Write Introduction

Missing figure

Insert senseful schematic picture/graph

The goal of this Student Research Paper is the implementation of a critical clearing time (CCT) determing Python algorithm for a single machine infinite bus (SMIB) model. Therefore a handful of faults or fault scenarios shall be simulated with the program. In combination with a few visualizations the concepts of transient stability assessment, and therefore determing the CCT and the critical power angle, is illustrated.

2 Fundamentals

Input of basic knowledge for system modelling; Maybe supplementary knowledge

[MK2]: Write Chap Fundamentals

General sources in terms of standard literature: [2]–[5]

2.1 Basics synchronous generators

- characteristics of a synchronous generator; structure and types of SG's
- mathematical background and description of the behavior -> dynamic modelling
- **Swing equations**

The final swing equation system can be derived to following two equations, which have to be solved in every time step to determine the pole angle δ and the rotor speed ω, respectively the rotor speed change from its base value $\Delta\omega$:

$$\frac{d\delta}{dt} = \Delta\omega \tag{2.1}$$

$$\frac{d\Delta\omega}{dt} = \frac{1}{2 \cdot H_{\mathrm{gen}}} \cdot (P_{\mathrm{m}} - P_{\mathrm{e}}) \tag{2.2}$$

where

δ	power angle
$\Delta\omega$	change of rotor angular speed
H_{gen}	inertia constant of the SG
P_{m}	mechanical power of the turbine
P_{e}	electrical power demanded transferred out of the SG

The generation of a time domain solution (TDS) for this equation system takes place in section 3.3.

2.2 System stability esp. transient context

- What is to be analyzed? And why? -> different stability analysis
- rotor angle stability,
- **derivation of EAC,**
- basic assessment models (single machine infinite bus, see [3])

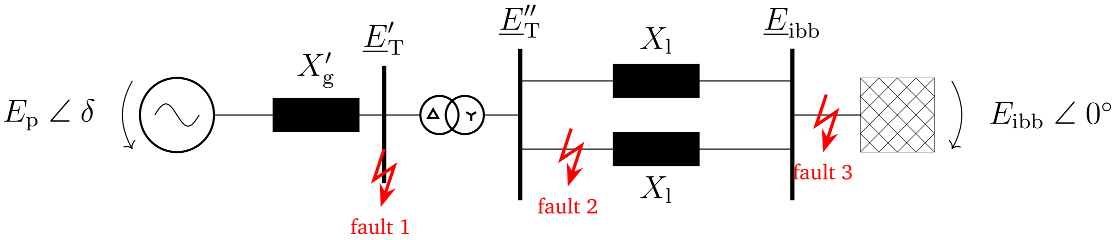

Figure 2.1: Representative circuit of a single machine infinite bus (SMIB) model with pole wheel voltage $E_p \angle \delta$ and infinite bus bar (IBB) voltage $E_{ibb} \angle 0°$; positions of considered faults 1 to 3 are marked with red lightning arrows

where

δ	power angle
E_p	pole potential of the synchronous generator (SG)
$\underline{E}'_T$	complex potential on the primary side of the transformer
$\underline{E}''_T$	complex potential on the secondary side of the transformer
$\underline{E}_{ibb}$	complex pole potential of the infinite bus bar (IBB)
X'_g	reactance of the synchronous generator (SG)
X_l	reactance of a single line

2.3 Events harming the system stability

- **fault types,**
- load-changes

- effects of electrical networks (esp. generator networks) vs. single machine systems -> paper [7]: IBB not that extremely fixed, group of critical and non-critical machines; but more of an outlook and targeting real-time calculation (for system operation)

2.4 Numerical methods for TDSs and system modeling

- **solving second order ODEs (explicit)**
- Differentiation explicit/impolicit, inertial value problems, boundary value problems, ...

System dynamics is a method for describing, understand, and discuss complex problems in the context of system theory **[SOURCE]**. They often can be described through a set of coupled ordinary differential equations (ODEs), most resoluted in time dimension. How to bridge towards different boundary types, explicit and implicit methods, ...; Different solving methods, ...

ODEs can be solved through numerical integration with different methods. An easy and less complex method is Euler's method. It uses a linear extrapolation to calculate the functions value at the next timestep, so following the iterable function

$$f_{\mathrm{t+1}} = f_{\mathrm{t}} + \left(\frac{df}{dt}\right)_{\mathrm{t}} \cdot \Delta t, \tag{2.3}$$

with t being the time and f an on t dependent function. Generally a system of second order ODEs can be rewritten as two first order equations. This often simplifies the calculation or the use of numerical methods. The presented swing equation of a SG in Equation 2.1 and Equation 2.2 has been split up by that principle.

3 Numerical modelling

Following chapter will describe the implementation of Python Code for solving the derived ODE system (see section 2.1). For this the Python version 3.9 was used, in combination with the packages scipy, numpy, and matplotlib.[1] The complete code is included in the Appendix A.

[MK3]: Write Chap Methods

3.1 Structure of the CCT assessment

Program plan for determination / algorithm structure, containing:

- Pre questions:
  1. What do I want to know from the algorithm?
  2. What do I want to see?
- Answers / Hints for the algorithm:
  1. What are needed inputs?
  2. What are needed functions?
  3. How do partial results interact with each other / puzzle together to the superior question?

[1] documentation and manual can be found on *https://scipy.org/* [8], similiar for *matplotlib*, and *numpy* packages

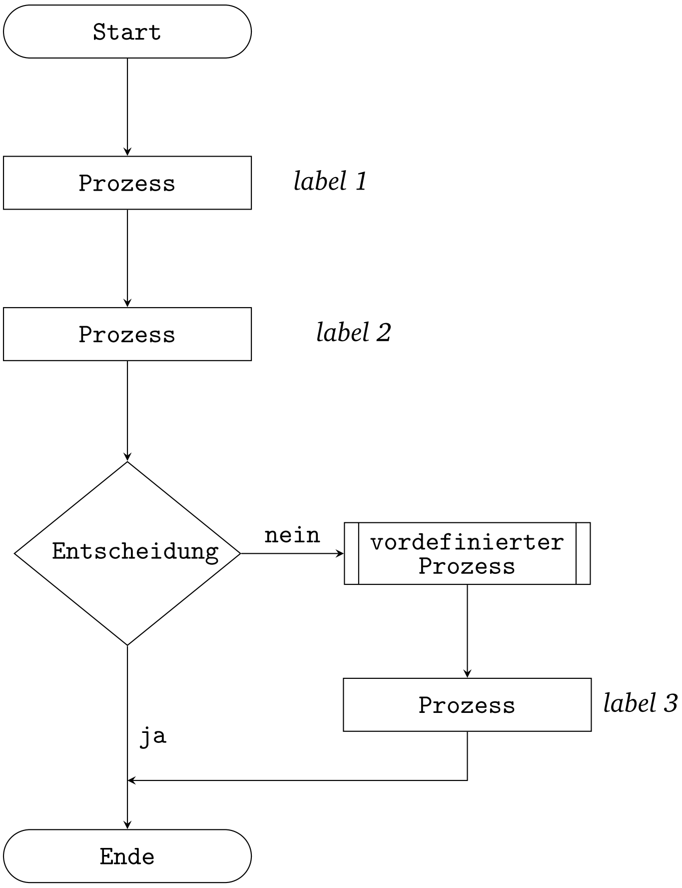

Figure 3.1: Program plan for determing the critical clearing time (CCT)

3.2 Electrical simplifications and scenario setting

- Simplification of all the components in SMIB network to a simple network
- Transforming into symmetrical components (for determination of shorts -> e.g. transformer)

Simplified networks, which are in interest and shall be simulated with the algorithm; ideally solved with subfigures

3.3 Implementation of the time domain solution

3.4 Implementation of the equal area criterion

3.5 Implementation of helping functions

4 Results

[MK4]: Write Chap Results

4.1 Analytical results

4.2 Numerical results

5 Discussion

[MK5]: Write Chap Discussion

6 Summary and outlook

Short summary of the results.

[MK6]: Write summary and outlook

A brief look in the future and why this topic is in the interest, maybe for slightly other applications as well (see [9]).

Acronyms

CCT critical clearing time
IBB infinite bus bar
ODE ordinary differential equation
SG synchronous generator
SMIB single machine infinite bus
TDS time domain solution

Symbols

Complete list of Symbols

H_{gen}	s	inertia constant of a synchronous generator (SG)
P	W	Power; electrical or mechanical

Bibliography

[1] "Perspektiven der elektrischen Energieübertragung in Deutschland," VDE Verband der Elektrotechnik Elektronik Informationstechnik e.V., Ed., Frankfurt am Main, Apr. 2019.

[2] J. D. Glover, T. J. Overbye, and M. S. Sarma, "Power system analysis & design," Boston, MA, 2017.

[3] P. S. Kundur and O. P. Malik, *Power System Stability and Control*, Second edition. New York Chicago San Francisco Athens London Madrid Mexico City Milan New Delhi Singapore Sydney Toronto: McGraw Hill, 2022, 948 pp., ISBN: 978-1-260-47354-4.

[4] J. Machowski, Z. Lubosny, J. W. Bialek, and J. R. Bumby, *Power System Dynamics: Stability and Control*, Third edition. Hoboken, NJ, USA: John Wiley, 2020, 1 p., ISBN: 978-1-119-52636-0 978-1-119-52638-4.

[5] D. Oeding and B. R. Oswald, *Elektrische Kraftwerke und Netze*, 8. Auflage. Berlin [Heidelberg]: Springer Vieweg, 2016, 1107 pp., ISBN: 978-3-662-52702-3. DOI: 10.1007/978-3-662-52703-0.

[6] A. J. Schwab, *Elektroenergiesysteme: smarte Stromversorgung im Zeitalter der Energiewende*, 7. Auflage. Berlin [Heidelberg]: Springer Vieweg, 2022, 871 pp., ISBN: 978-3-662-64773-8.

[7] S. Batchu, Y. Raghuvamsi, and K. Teeparthi, "A Comparative Study on Equal Area Criterion Based Methods for Transient Stability Assessment in Power Systems," in *2022 22nd National Power Systems Conference (NPSC)*, New Delhi, India: IEEE, Dec. 17, 2022, pp. 124–129, ISBN: 978-1-66546-202-0. DOI: 10.1109/NPSC57038.2022.10069303. [Online]. Available: https://ieeexplore.ieee.org/document/10069303/ (visited on 12/14/2023).

[8] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2, 2020, ISSN: 1548-7091, 1548-7105. DOI: 10.1038/s41592-019-0686-2. [Online]. Available: https://www.nature.com/articles/s41592-019-0686-2 (visited on 01/13/2024).

[9] Z. Gao, W. Du, and H. Wang, "Transient stability analysis of a grid-connected type-4 wind turbine with grid-forming control during the fault," *International Journal of Electrical Power & Energy Systems*, vol. 155, p. 109 514, Jan. 2024, ISSN: 01420615. DOI: `10.1016/j.ijepes.2023.109514`. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/S0142061523005719` (visited on 12/15/2023).

Appendix

A Code

A.1 Model functions

A.2 Model of GK

```
import matplotlib.pyplot as plt
import numpy as np


def mag_and_angle_to_cmplx(mag, angle):
    return mag * np.exp(1j * angle)


fn = 60

H_gen = 3.5
X_gen = 0.2
X_ibb = 0.1
X_line = 0.65

# Values are initialized from loadflow
E_fd_gen = 1.075
E_fd_ibb = 1.033
P_m_gen = 1998/2200

omega_gen_init = 0
delta_gen_init = np.deg2rad(45.9)
delta_ibb_init = np.deg2rad(-5.0)

v_bb_gen_init = mag_and_angle_to_cmplx(1.0, np.deg2rad(36.172))


def differential(omega, v_bb_gen, delta):
    # Calculate the electrical power extracted from the generator at its busbar.
    E_gen_cmplx = mag_and_angle_to_cmplx(E_fd_gen, delta)
    P_e_gen = (v_bb_gen * np.conj((E_gen_cmplx - v_bb_gen) / (1j * X_gen))).real

    # transform the constant mechanical energy into torque
    T_m_gen = P_m_gen / (1 + omega)

    # Differential equations of a generator according to Machowski
    domega_dt = 1 / (2 * H_gen) * (T_m_gen - P_e_gen)
    ddelta_dt = omega * 2 * np.pi * fn

    return domega_dt, ddelta_dt


def algebraic(delta_gen, sc_on):
    # If the SC is on, the admittance matrix is different.
```

```
    # The SC on busbar 0 is expressed in the admittance matrix as a very large
        admittance (1000000) i.e. a very small impedance.
    if sc_on:
        y_adm = np.array([[(-1j / X_gen - 1j / X_line) + 1000000, 1j / X_line],
                          [1j / X_line, -1j / X_line - 1j / X_ibb]])
    else:
        y_adm = np.array([[-1j / X_gen - 1j / X_line, 1j / X_line],
                          [1j / X_line, -1j / X_line - 1j / X_ibb]])

    # Calculate the inverse of the admittance matrix (Y^-1)
    y_inv = np.linalg.inv(y_adm)

    # Calculate current injections of the generator and the infinite busbar
    i_inj_gen = mag_and_angle_to_cmplx(E_fd_gen, delta_gen) / (1j * X_gen)
    i_inj_ibb = mag_and_angle_to_cmplx(E_fd_ibb, delta_ibb_init) / (1j * X_ibb)

    # Calculate voltages at the bus by multiplying the inverse of the admittance
        matrix with the current injections
    v_bb_gen = y_inv[0, 0] * i_inj_gen + y_inv[0, 1] * i_inj_ibb
    v_bb_ibb = y_inv[1, 0] * i_inj_gen + y_inv[1, 1] * i_inj_ibb

    return v_bb_gen


def do_sim():

    # Initialize the variables
    omega_gen = omega_gen_init
    delta_gen = delta_gen_init
    v_bb_gen = v_bb_gen_init

    # Define time. Here, the time step is 0.005 s and the simulation is 5 s long
    t = np.arange(0, 5, 0.005)
    x_result = []

    for timestep in t:

        # Those lines cause a short circuit at t = 1 s until t = 1.05 s
        if 1 <= timestep < 1.05:
            sc_on = True
        else:
            sc_on = False

        # Calculate the initial guess for the next step by executing the
            differential equations at the current step
        domega_dt_guess, ddelta_dt_guess = differential(omega_gen, v_bb_gen,
            delta_gen)
        omega_guess = omega_gen + domega_dt_guess * (t[1] - t[0])
        delta_guess = delta_gen + ddelta_dt_guess * (t[1] - t[0])

        v_bb_gen = algebraic(delta_guess, sc_on)

        # Calculate the differential equations with the initial guess
        domega_dt_guess2, ddelta_dt_guess2 = differential(omega_guess, v_bb_gen,
            delta_guess)

        domega_dt = (domega_dt_guess + domega_dt_guess2) / 2
        ddelta_dt = (ddelta_dt_guess + ddelta_dt_guess2) / 2
```

```python
        omega_gen = omega_gen + domega_dt * (t[1] - t[0])
        delta_gen = delta_gen + ddelta_dt * (t[1] - t[0])

        v_bb_gen = algebraic(delta_gen, sc_on)


        # Save the results, so they can be plotted later
        x_result.append(omega_gen)

    # Convert the results to a numpy array
    res = np.vstack(x_result)
    return t, res


if __name__ == '__main__':

    # Here the simulation is executed and the timesteps and corresponding results are returned.
    # In this example, the results are omega, delta, e_q_t, e_d_t, e_q_st, e_d_st of the generator and the IBB
    t_sim, res = do_sim()

    # load the results from powerfactory for comparison
    delta_omega_pf = np.loadtxt('pictures/powerfactory_data.csv', skiprows=1, delimiter=',')

    # Plot the results
    plt.plot(t_sim, res[:, 0].real, label='delta_omega_gen_python')
    plt.plot(delta_omega_pf[:, 0], delta_omega_pf[:, 1] - 1, label='delta_omega_gen_powerfactory')
    plt.legend()
    plt.title('Reaction of a generator to a short circuit')

    plt.savefig('pictures/short_circuit_improved.png')

    plt.show()
```

Listing A.1: GK's SMIB model with Heun's integration method

B Additional

B.1 Jupyter notebook for development