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

Insert some fancy introduction

1 **Introduction (1 page)**

2 **State-of-the-art (~ 4 pages)**

  2.1 Basics synchronous generators
  -> **swing-equations**

  2.2 System stability esp. transient context
  -> rotor angle stability, **derivation of EAC,** basic assessment models (single machine infinite bus, see [1])

  2.3 Numerical methods for TDSs and system modelling
  -> **solving second order ODEs (explicit)**

  2.4 Events harming the system stability
  -> **faults,** load-changes, effects of electrical networks (esp. generator networks) vs. single machine systems

3 **Numerical modeling (~ 5 pages)**

  3.1 (Object relations and classes)

  3.2 Algorithm and functional structure

  3.3 Implementation of functions and dependencies

  3.4 Implementation of numerical solvers

4 **Results (~ 3 pages)**

  4.1 Analytical results

  4.2 Numerical results

5 **Discussion (~ 2 pages)**

  5.1 Numerical vs. analytical

  5.2 (Single machine vs. network models)

  5.3 ... (dependent on time and outcomes)

6 **Summary and outlook (1 page)**

Total amount ~ 16 pages (without appendix and supplementary pages)

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

Das ist ein Testkommentar.

Introduction via [2] and other standard literature like [1], [3]–[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; .

2 Fundamentals

Input of basic knowledge for system modelling; Maybe supplementary knowledge

General sources in terms of standard literature: [1], [3]–[5]

2.1 Basics synchronous generators

- **Swing equations**
- Characteristics of a synchronous generator
- types of SG's

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 [1])

2.3 Numerical methods for TDSs and system modeling

- **solving second order ODEs (explicit)**

2.4 Events harming the system stability

- **fault types,**
- load-changes
- effects of electrical networks (esp. generator networks) vs. single machine systems

3 Numerical modelling

3.1 Algorithm and functional structure

Describing the basic functionality and compartements of the model.

3.2 Implementation of functions and dependencies

Describing implementation into Python-code.

3.3 Implementation of numerical solvers

Describing the functionality and structure of (excplicit) numerical methods. Starting from Euler (basic) to a more complex but more reliable method (Heun, predictor-corrector, ...). Main focus: Implementation into Python.

Euler's method

Heun's method

Heun's method is implemented in Python. An example is provided in Listing A.2

4 Results

4.1 Analytical results

4.2 Numerical results

5 Discussion

5.1 Analytical vs. numerical

5.2 Single machine vs. network models

6 Summary and outlook

In der Zusammenfassung werden die Ergebnisse der Arbeit kurz zusammengefasst. Der Umfang beträgt ca. eine Seite.

Acronyms

TDS time domain solution

Bibliography

[1] 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.

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

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

[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.

Appendix

A Code

A.1 Model functions

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

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

# Define the parameters of the system
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

# init states of variables
omega_gen_init = 0 # init state
delta_gen_init = np.deg2rad(45.9) # init state
delta_ibb_init = np.deg2rad(-5.0) # init state

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

result_ode = []


def smib_model(result_ode, t):
    # defines a ode 2nd order ode for decribing the dynamic behavior of a synchronous generator vs. an infinite bus
    # Those lines cause a short circuit at t = 1 s until t = 1.05 s
    if 1 <= t < 1.1001:
        sc_on = True
    else:
        sc_on = False

    # 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

    # 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 [result_ode[0], result_ode[1]] # domega_dt, ddelta_dt

if __name__ == "__main__":
    def showplot():
        from matplotlib import pyplot as plt
        x = [1,5,10,15]
        y = [12,59,100,155]

        plt.plot(x, y)
        plt.show()
```

Listing A.1: Module containing all relevant functions of the SMIB model in Python

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 = []
```

```python
    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

        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.2: GK's SMIB model with Heun's integration method