LAPORAN TUGAS BESAR METODE NUMERIS: ANALISIS HUBUNGAN KUAT TORSI TERHADAP PANAS DARI MAGNET MOTOR ELEKTRIK

Disusun oleh :
YITZHAK EDMUND TIO MANALU
(22/499769/TK/54763)

PROGRAM STUDI S-1 TEKNOLOGI INFORMASI
UNIVERSITAS GADJAH MADA
TAHUN AJARAN 2023/2024

A. PENDAHULUAN

I. Permasalahan yang Diangkat

Permasalahan yang diangkat pada laporan ini berkaitan dengan hubungan perubahan panas magnet motor elektrik terhadap perubahan kuat torsi yang dimiliki oleh motor elektrik tersebut. Adapun data yang dimiliki terdiri dari 150 pasang nilai torsi dan panas magnet yang perlu diformulasikan terlebih dahulu untuk mengetahui karakteristiknya.

II. Tujuan Simulasi Numeris

Tujuan dari perlakuan simulasi numeris ini di antaranya adalah,

a. Mengetahui hubungan antara nilai torsi dan panas pada magnet motor elektrik
b. Mengetahui saat di mana nilai panas pada magnet motor elektrik bermula atau berhenti

III. Kekangan Simulasi

Simulasi ini belum menggunakan kekangan sebagai pertimbangan tambahan dalam formulasinya.

B. METODE

I. Metode Numeris yang Digunakan beserta Alasannya

Tujuan dari perlakuan simulasi numeris ini di antaranya adalah untuk menggabungkan ketiga metode numeris utama yang diperlukan, yaitu Gauss-Seidel, Newton Raphson, dan Secant Method. Oleh karena itu, untuk menghubungkan ketiga metode numeris, perlu diformulasikan alur logika yang tepat.

Metode numeris Gauss-Seidel merupakan metode yang pada dasarnya berfungsi untuk menyelesaikan persamaan linear. Selain itu, metode Newton Raphson dan Secant Method memiliki fungsi yang sama sehingga dapat dianggap sebagai pembanding antara satu sama lain saja. Tetapi, perlu diketahui bahwa terdapat perbedaan masukan dan keluaran antara kedua jenis metode ini.

Gauss-Seidel merupakan metode numeris bertipe Linear Equation Solver yang menerima input berupa matriks dan vektor dari persamaan linear dengan output berupa vektor yang berisikan nilai-nilai solusi dari persamaan linear. Hal ini berbeda dengan Newton Raphson dan Secant Method yang merupakan metode numeris bertipe Root-Finding. Kedua metode ini menerima input berupa fungsi dan mengeluarkan akar-akar dari fungsi yang diberikan. Oleh karena itu, diperlukan penghubung antara kedua jenis metode ini untuk menyelesaikan masalah yang diberikan dengan penambahan metode Power Linear Regression.

Metode Power Linear Regression berfungsi untuk mengolah data mentah menjadi regresi linear berpangkat. Regresi linear berpangkat ini memerlukan tahap penyelesaian persamaan linear pada bagiannya. Oleh karena itu, Power Linear Regression yang diperkuat dengan Gauss-Seidel digunakan untuk mengubah dataset menjadi sebuah persamaan linear. Kemudian, persamaan linear inilah yang diteruskan kepada Root-Finding methods, yaitu Newton Raphson dan Secant Method untuk ditemukan akar-akarnya (titik awal ketika y = 0 atau ketika fungsi berhasil turun kembali menyentuh sumbu y = 0).

Secara garis besar, berikut adalah alur metode numeris yang digunakan:

1. Data Set ➔ [Power Linear Regression] ➔ [Gauss-Seidel] ➔ Function
2. Function ➔ [Newton Raphson & Secant Method] ➔ Roots

II. Estimasi Awal

Estimasi awal untuk bagian metode Gauss-Seidel tidak dispesifikasikan, sehingga default value pada fungsi langsung dijalankan, yaitu dengan menganggap estimasi = zero vector. Selain itu, estimasi tidak diperlukan untuk bagian metode numeris yang lainnya.

III. Kriteria Pemberhentian Simulasi

Simulasi diberhentikan dengan berbagai macam pertimbangan. Karena program menerapkan 4 jenis metode numeris, yaitu Power Linear Regression, Gauss-Seidel, Newton Raphson, dan Secant Method, diperlukan kriteria yang dapat mendukung kebutuhan masing-masing metode untuk melakukan pemberhentian simulasi.

Pada metode Power Linear Regression, tidak ada konfigurasi khusus untuk limitasi, kecuali penentuan pangkat tertinggi dengan pangkat tertinggi = regressPower – 1. Selain itu, pada metode Gauss-Seidel, limitasi dilakukan kepada nilai toleransi error, jumlah nilai tertoleransi minimum yang diizinkan, dan limit maksimal iterasi. Terakhir, pada kedua metode Root-Finding yang digunakan, hanya diterapkan nilai toleransi error dan limit maksimal iterasi yang sama nilainya seperti pada limitasi metode Gauss-Seidel.

Berikut adalah cuplikan konfigurasi yang terdapat pada kode yang dapat dikembangkan menjadi customizable.

```
# Setting Up Configurations
# Method Setting
regressPower = 4    # Regressor Power
gseidMinErrPass = 3
# Minimum vector values that pass the error tolerance ex. for val = 2, [0%, 0%, 5%] is considered to as valid to return

# Input Guess
inputBefVal = 0
inputVal = 0.7
inputGuess = []  # n = Regressor power: otherwise, return error
# [alternative: leave as an empty array to automatically make zero vector guess]

# Import Value
val_x = x_vector
val_y = y_vector
funct = sympify("x**2 - 1")

matrixIn = inMatrix
vectorIn = inVector

# Round and Tolerance
inputRound = 50
errRound = 10
errTolerance = 0.001
iterLimit = 1000000

# View Toggle
viewProcess = False
```

IV. Pseudocode

(masih dalam tahap formulasi)

V. Code

(masih terdapat beberapa fitur yang dalam tahap pengembangan)

(Apabila hendak mencoba, sebaiknya lakukan input angka 1 terus menerus hingga output keluar karena *user interface* yang dikembangkan masih belum sempurna)

```
# Importing Packages
import copy
import os
from sympy import *
from InputX import x_vector
from InputY import y_vector
from InputGauss import inMatrix, inVector
```

```
# Creating Methods
x = symbols('x')

# Evaluate X-Input


def evToX(f, val):
    return N(f.subs(x, val))

# Array Printer


def printArray(arr, roundV, postfix=""):
    print("[", end="")
    for i in range(len(arr)):
        if (i != 0):
            print(", ", end="")
        print(str(round(arr[i], roundV))+postfix, end="")
    print("]", end="")

# Array to Function


def arrToFunc(vect):
    stringify = ""
    for i in range(len(vect)):
        # Extract values
        coef = str(vect[i])
        powr = "x**" + str(i)
        stringed = ""

        # Combine per operator
        if not (i == 0):
            if not (coef[0] == "-"):
                stringed += " +"
            else:
                stringed += " "

        # Combine per operand
        stringed += (coef + "*" + powr)

        # Export
        stringify += stringed

    # Print Test
    # print("String function:", stringify)
    funct = sympify(stringify)
```

```
    return funct

# NewtonRaphson


def NewtonRaphson(f, f_dif, val, roundV, errRound, tolerance=0.00001,
limit=1000, it=0, errPrev=0, view=True):
    # Intializing Process
    if (it == 0):
        print("Start to evaluate Newton Raphson")

    # Simulation Terminator: Limit of Loop
    if (it == limit):
        print(
            "Iteration Limit! The function is either divergent or requires
more iteration!")
        return [val, it]

    # Newton Raphson Formula + Error Evaluation
    value = val - evToX(f, val) / evToX(f_dif, val)
    error = (value - val)/value

    if view:
        print("Iterate: ", it+1, "\tPrevious: ", val, "\tCurrent: ",
              value, "\tError: ", round(error*100, errRound), "%")

    # Updating Process
    iter = it + 1
    terminate = False

    # Simulation Terminator: Found --- 0 Error a.k.a value is found
    if (error == 0):
        print("Stopped by the exact value found")
        terminate = True

    # Simulation Terminator: Error Tolerance --- Error difference is too small
to continue
    elif (it != 0 and abs(error - errPrev) < tolerance):
        print("Stopped by the error tolerance limit")
        terminate = True

    # Simulation Terminator: Almost exact --- Change of value is too small to
continue
    elif (round(val, roundV) == round(value, roundV)):
        print("Stopped by the tolerated round value")
        terminate = True

    # Terminator
    if terminate:
```

```
        print("Newton Raphson Stopped! Last Value:", value)
        return [val, it]

    return NewtonRaphson(f, f_dif, value, roundV, errRound,
                         tolerance, limit, iter, error, view)

# Secant Metho


def SecantMethod(f, valCurr, valBef, roundV, errRound, tolerance=0.00001,
limit=1000, it=0, errPrev=0, view=True):
    # Intializing Process
    if (it == 0):
        print("Start to evaluate Secant Method")

    # Simulation Terminator: Limit of Loop
    if (it == limit):
        print(
            "Iteration Limit! The function is either divergent or requires
more iteration!")
        return [round(valCurr, roundV), it]

    # Secant Method Formula + Error Evaluation
    valUpd = valCurr - (
        evToX(f, valCurr) * (
            valBef - valCurr
        )
    ) / (
        evToX(f, valBef) - evToX(f, valCurr)
    )
    error = (valUpd - valCurr)/valUpd

    if view:
        print("Iterate: ", it+1, "\tPrevious: ", valCurr, "\tCurrent: ",
              valUpd, "\tError: ", round(error*100, errRound), "%")

    # Updating Process
    iter = it + 1
    terminate = False

    # Simulation Terminator: Found --- 0 Error a.k.a value is found
    if (error == 0):
        print("Stopped by the exact value found")
        terminate = True

    # Simulation Terminator: Error Tolerance --- Error difference is too small
to continue
    elif (it != 0 and abs(error - errPrev) < tolerance):
        print("Stopped by the error tolerance limit")
```

```
        terminate = True

    # Simulation Terminator: Almost exact --- Change of value is too small to
continue
    elif (round(valCurr, roundV) == round(valUpd, roundV)):
        print("Stopped by the tolerated round value")
        terminate = True

    # Terminator
    if terminate:
        print("Secant Method Stopped!  Last Value:", valUpd)
        return [round(valUpd, roundV), it]

    try:
        return SecantMethod(f, valUpd, valCurr, roundV, errRound, tolerance,
limit, iter, error, view)
    except:
        print("Exception Error")
        return ["Failed", "Failed"]


# Gauss-Seidel
def GaussSeidel(roundV, errTol, totalTerm, b_val, matrix, guess=0, limit=100,
view=True):
    # Exception Handler
    if (guess != 0 and (len(guess) != len(b_val))):
        print("Inequal length of 'guess-vector' and 'y-vector'\n",
              "Terminating the whole process!", sep="")
        return [0, 0]

    # Init Message
    print("Start to Evaluate Gauss Seidel Method")

    # Init Values
    iteration = 0
    arrOut = []
    errEnd = []

    errX = []
    x_valProcess = []

    # Guess-Handler
    if (guess == 0):  # If it is not set, set all to zero
        for i in range(len(b_val)):
            errX.append(0)
        x_valProcess = copy.deepcopy(errX)
    else:  # If it is set, copy it to process and error comparison
        x_valProcess = guess
        errX = copy.deepcopy(guess)
```

```
    # Loop the Gauss-Seidel
    for k in range(limit):
        # Terminate Message
        termMsg = ""

        if view:
            print("Loop :", k+1, "\nBefore: ", end="")

            # Current value Display
            printArray(x_valProcess, roundV)

            print()

        # Single Gauss-Seidel Process
        prev_x_valProcess = copy.deepcopy(x_valProcess)
        for j in range(len(matrix)):
            val = 0
            # Adding value to the b-value
            # print("Add", b[j])
            val += b_val[j]

            for i in range(len(matrix[j])-1):
                index = (j + 1 + i) % len(matrix[j])
                # Subtracting the value to x vars with response to the DYNAMIC
x-input
                # print(-vars[j][index], "x" + str(index+1))
                val -= matrix[j][index] * x_valProcess[index]

            # Dividing the value to the analyzed x-output
            # print("Div by", vars[j][j])
            val /= matrix[j][j]

            # Move value to the storage
            x_valProcess[j] = val

        # Error Calculation
        triggerTerminate = False

        # Error Tolerance Count
        toleratedError = 0
        zeroError = 0

        if view:
            print("Current: ", end="")

            # Current value Display
            printArray(x_valProcess, roundV)
```

```
            print()

        for i in range(len(errX)):
            errX[i] = abs((x_valProcess[i] - prev_x_valProcess[i]) /
                          x_valProcess[i])*100

            # Terminate if the minimum error achieved
            if (abs(errX[i]) < errTol):
                toleratedError += 1
                if ((not triggerTerminate) and (toleratedError == totalTerm)):
                    termMsg = "Terminated by tolerated error value"
                    triggerTerminate = True

            if (errX[i] == 0):
                zeroError += 1
                if ((not triggerTerminate) and (zeroError == totalTerm)):
                    termMsg = "Terminated by zero error achieved"
                    triggerTerminate = True

        if view:
            print("Error: ", end="")
            printArray(errX, roundV, postfix="%")
            print("\n")

        # Stop if there exist tolerable error
        if (triggerTerminate):
            # Save the Result
            iteration = k+1
            arrOut = x_valProcess
            errEnd = errX
            break

        # Save if Limit
        if (k == limit - 1):
            termMsg = "Terminated by Limit of iteration"
            iteration = k+1
            arrOut = x_valProcess
            errEnd = errX

    print(termMsg)
    print("Result: ", end="")
    printArray(x_valProcess, roundV)
    print()
    print("Error : ", end="")
    printArray(errEnd, roundV, postfix="%")
    print()

    return [arrOut, iteration]
```

```python
# Power Regressor, Array Formulator
def powerRegressor(x_val=[], y_val=[], power=1):
    # Exception Handler
    if len(x_val) != len(y_val):
        print("Invalid input! Cannot regress different size of x_val and
y_val!")
        return

    # Evaluate Matrix
    # Take the n
    n_val = len(x_val)  # Bascially the same as len(y_val)
    matrix = []

    # Iterate through matrix
    for i in range(power):
        vector = []

        # Iterate through vector
        for j in range(power):
            # Only append if it's not the very first element
            if (i == 0 and j == 0):
                vector.append(n_val)
                continue

            # Iterate through sum
            sum = 0
            for k in range(len(x_val)):
                # Sigma of x_val^(row+col)
                sum += x_val[k]**(i+j)

            vector.append(sum)

        matrix.append(vector)

    # Test
    # print("Matrix:")
    # for i in range(len(matrix)):
    #     for j in range(len(matrix[i])):
    #         print(matrix[i][j], end="\t\t")
    #     print()

    # Evaluate Res_Vector
    res_vector = []

    # Iterate through y
    for i in range(power):
        sum = 0
```

```
        # Sum each power
        for j in range(len(y_val)):
            # Component y & x
            y_comp = y_val[j]
            x_comp = x_val[j]**(i)
            # Sum
            sum += y_comp * x_comp
        # Append vector
        res_vector.append(sum)

    # Test
    # print("Vector:")
    # print(res_vector)

    return [matrix, res_vector]


# Setting Up Configurations
# Method Setting
regressPower = 4    # Regressor Power
gseidMinErrPass = 3
# Minimum vector values that pass the error tolerance ex. for val = 2, [0%,
0%, 5%] is considered to as valid to return

# Input Guess
inputBefVal = 0
inputVal = 0.7
inputGuess = []  # n = Regressor power: otherwise, return error
# [alternative: leave as an empty array to automatically make zero vector
guess]

# Import Value
val_x = x_vector
val_y = y_vector
funct = sympify("x**2 - 1")

matrixIn = inMatrix
vectorIn = inVector

# Round and Tolerance
inputRound = 50
errRound = 10
errTolerance = 0.001
iterLimit = 1000000

# View Toggle
viewProcess = False

# Execute Function
```

```
def PowRegProcess():
    # Calling Methods for Power Regressor
    matrixPR, vectorPR = powerRegressor(
        x_val=(val_x), y_val=(val_y), power=(regressPower))

    # print(matrixPR)
    # print(vectorPR)

    return [matrixPR, vectorPR]


def GauSedProcess(matrixPR, vectorPR, convertFunc=True):
    # Calling Methods for Linear Algebra Solving
    gaussSeidRes, gaussSeidIter = GaussSeidel(roundV=(inputRound),
errTol=(errTolerance), totalTerm=(
        gseidMinErrPass), b_val=(vectorPR), guess=(inputGuess if inputGuess
else 0), matrix=(matrixPR), limit=(iterLimit), view=(viewProcess))

    if gaussSeidIter == 0:
        return
    print()

    # Convert Result to Function
    funcYes = 0
    if convertFunc:
        funcYes = arrToFunc(gaussSeidRes)

    return [funcYes, gaussSeidRes, gaussSeidIter]


def SrRootProcess(funcYes):
    # Calling Methods Root-Finding
    newtRaphRes, newtRaphIter = NewtonRaphson(f=(funcYes),
f_dif=(diff(funcYes, x)), val=(inputVal), roundV=(
        inputRound), errRound=(errRound), tolerance=(errTolerance),
limit=(iterLimit), view=(viewProcess))
    print()
    secnMthdRes, secnMthdIter = SecantMethod(valCurr=(inputVal),
valBef=(inputBefVal), roundV=(
        inputRound), errRound=(errRound), tolerance=(errTolerance),
f=(funcYes), limit=(iterLimit), view=(viewProcess))
    print()

    return [newtRaphRes, newtRaphIter, secnMthdRes, secnMthdIter]
```

```
def analysis(gaussSeidRes, gaussSeidIter, funcYes, newtRaphRes, newtRaphIter,
secnMthdRes, secnMthdIter, isRootFinding, isLinEqSlving, isEqRegressor):
    # Making Up Conclusions:
    if (isEqRegressor):
        print("[Evaluation finished with the result]\n")
        print(f"Analyzing the dataset of:\n",
              f"x = {val_x}\n",
              f"y = {val_y}\n")
        print("Analysis of Regressor:")

    if (isLinEqSlving or isEqRegressor):
        print(
            f"Gauss  Seidel  resulted in the vector of:\n{gaussSeidRes}\nwith
{gaussSeidIter} iterations.\n")
        if (isEqRegressor):
            print(
                f"Translated into Power Regressor in a function
of:\n{funcYes}", end="\n\n")

    if (isRootFinding):
        print("Analysis of Root:")
        print(
            f"Newton Raphson resulted in:\n{newtRaphRes}\nwith {newtRaphIter}
iterations.")
        print(
            f"Secant Method  resulted in:\n{secnMthdRes}\nwith {secnMthdIter}
iterations.")
        print("\n[Evaluation completed]")


def exec(isRootFinding, isLinEqSlving, isEqRegressor):
    # Calling PowerReg
    matrixPR = []
    vectorPR = []
    if (isEqRegressor):
        matrix, vector = PowRegProcess()
        matrixPR = copy.deepcopy(matrix)
        vectorPR = copy.deepcopy(vector)

    # Calling GaussSed
    funcYes = funct
    gaussSeidRes = []
    gaussSeidIter = 0
    if (isLinEqSlving or isEqRegressor):
        if (isEqRegressor):
            func, gaussSdRes, gaussSdIter = GauSedProcess(matrixPR, vectorPR)
        else:
            func, gaussSdRes, gaussSdIter = GauSedProcess(
                matrixIn, vectorIn, convertFunc=(False))
```

```
        funcYes = func
        gaussSeidRes = copy.deepcopy(gaussSdRes)
        gaussSeidIter = gaussSdIter

    # Calling RootFind
    newtRaphRes = []
    newtRaphIter = 0
    secnMthdRes = []
    secnMthdIter = 0

    if (isRootFinding):
        newtRhRes, newtRIter, secnMRes, secnMIter = SrRootProcess(funcYes)
        newtRaphRes = copy.deepcopy(newtRhRes)
        newtRaphIter = newtRIter
        secnMthdRes = copy.deepcopy(secnMRes)
        secnMthdIter = secnMIter

    analysis(funcYes=(funcYes), gaussSeidIter=(gaussSeidIter),
gaussSeidRes=(gaussSeidRes), newtRaphIter=(
        newtRaphIter), newtRaphRes=(newtRaphRes), secnMthdIter=(secnMthdIter),
secnMthdRes=(secnMthdRes), isRootFinding=(isRootFinding),
isLinEqSlving=(isLinEqSlving), isEqRegressor=(isEqRegressor))

# User Interface


def u_interface_select():
    return input("Choose: ")


def u_interface_wlcm():  # Welcome
    print("Welcome to Numerical Method Analysis Program")


options_type = [
    "Please choose your Numerical Method Type:",
    "Root-Finding",
    "Linear Equation Solver",
    "Equation Regressor"
]

options_input = [
    "Pleace choose your input:",
    "From 'InputX' and 'InputY' File",
    "From manual input"
]
```

```
def optionHandler(options=[], taken=[], chosen=""):  # Reduce Options
    selected = copy.deepcopy(taken)

    # Check selection validity
    if (chosen != "") and (chosen in options):
        selected.append(chosen)
        options.remove(chosen)
    else:
        # Choice is either invalid or the options are empty.
        print("Invalid request!")
        return [False, False]

    # Test
    # print("Opts: ", options, "\nSlct: ", selected)

    return [options, selected]


def optionDisplayer(options=[], selected=[], multiChoice=True):  # Display
Options
    if len(options) <= 1:
        print("Cannot proceed to the next process as there is no option
available!")
        return [False, False]

    localOpts = copy.deepcopy(options)
    localSlct = copy.deepcopy(selected)

    while True:
        # Handle Interface Output
        print(localOpts[0])
        for i in range(len(localOpts)-1):
            print(i+1, ") ", localOpts[i+1], sep="")

        if localSlct != []:
            print("x) Finish Selection")

        # Input Handler
        inputVal = u_interface_select()

        if (localSlct != [] and inputVal == "x"):
            break

        try:  # Conversion Handler
            inputVal = int(inputVal)
        except:
            os.system('cls')
            print("Your input is not a number! Please try again!")
            continue
```

```
        if inputVal <= 0 or inputVal >= len(localOpts):  # Range Handler
            os.system('cls')
            print("Your input is out of range! Please try again!")
            continue

        # Process Input
        handledOpts, handledSelect = optionHandler(
            options=(localOpts), taken=(localSlct),
chosen=(localOpts[inputVal]))  # Update Options

        # Save Input
        localOpts = handledOpts
        localSlct = handledSelect

        os.system('cls')

        # Break if not multiChoice
        if not multiChoice:
            break

        # Break if no options left
        if len(localOpts) <= 1:
            break

    os.system('cls')
    return [handledOpts, handledSelect]


def u_interface_mthd(callback_input):  # Numerical Method Type
    options, selected = optionDisplayer(options_type)

    # Test
    # print("Opts", options)
    # print("Sclt", selected)

    # Booleans
    isRootFinding = options_type[1] in selected
    isLinEqSlving = options_type[2] in selected
    isEqRegressor = options_type[3] in selected

    manualInput = callback_input()

    # Selectors
    if (isRootFinding and isLinEqSlving and isEqRegressor):
        print(
            "You're choosing: Data Set -> Power Regression --[GS]--> Function
--[NR, SM]--> Root")
    elif (isRootFinding and isLinEqSlving):
```

```
        print("You're choosing: Matrix --[GS]--> Function --[NR, SM]--> Root")
    elif (isRootFinding and isEqRegressor):
        print("You're choosing: (there is no supporting method for this one)")
    elif (isLinEqSlving and isEqRegressor):
        print(
            "You're choosing: Dataset --> Power Regression --[GS]-->
Function")
    elif (isRootFinding):
        print("You're choosing: Root Finding [NR, SM]")
    elif (isLinEqSlving):
        print("You're choosing: Linear Equation Solving [GS]")
    elif (isEqRegressor):
        print("You're choosing: Equation Regression [Power+GS]")
    else:
        print("ERROR IN METHOD CHOICE SELECTION")

    print()

    exec(isRootFinding=(isRootFinding), isLinEqSlving=(
        isLinEqSlving), isEqRegressor=(isEqRegressor))


def u_interface_src():  # Data Source
    options, selected = optionDisplayer(options_input, multiChoice=(False))

    # Booleans
    isFromFile = options_input[1] in selected
    isManualIn = options_input[2] in selected

    # Selectors
    if (isFromFile):
        print("You're choosing: Exported Input")
        return False
    elif (isManualIn):
        print("You're choosing: Manual Input")
        return True
    else:
        print("ERROR IN SOURCE CHOICE SELECTION")

    print()


def u_interface():
    u_interface_wlcm()
    u_interface_mthd(callback_input=(u_interface_src))


# Execute
u_interface()
```

Kode memiliki *dependency* kepada *source code* yang berperan sebagai sumber masukan berikut:

InputX.py

```
x_vector = [
    79.99798584,
    80.31549835,
    79.98143005,
    80.2304306,
    9.336613655,
    22.78134727,
    -0.435703099,
    80.13986969,
    95.24025726,
    39.72328949,
    57.55591202,
    97.73615265,
    91.76512146,
    100.7842178,
    0.234816223,
    80.15705109,
    70.72084045,
    0.222863987,
    86.91890717,
    99.51862335,
    0.26093331,
    0.268791944,
    0.258909851,
    0.244049057,
    101.6950684,
    0.245598003,
    0.24124454,
    0.215281159,
    0.250325888,
    0.229329497,
    102.3352432,
    103.1373978,
    103.3696518,
    102.8179016,
    0.21569474,
    103.539772,
    104.0086136,
    0.19908613,
    0.231475934,
    103.975441,
    103.9572525,
    104.0030594,
```

InputY.py

```
y_vector = [
    20.85695648,
    20.8785553,
    20.88642883,
    20.89416122,
    21.03241158,
    21.03477478,
    21.0363884,
    21.03696251,
    21.04045105,
    21.041996,
    21.04244804,
    21.04585266,
    21.0470562,
    21.0471611,
    21.04819489,
    21.0484333,
    21.04846764,
    21.04847908,
    21.04873276,
    21.04965401,
    21.05058479,
    21.05484009,
    21.0555172,
    21.05767822,
    21.06048584,
    21.06051636,
    21.06550026,
    21.06569672,
    21.06653786,
    21.06693649,
    21.06896591,
    21.07456779,
    21.07531548,
    21.07788086,
    21.08103943,
    21.08125114,
    21.08442879,
    21.08604431,
    21.08892822,
    21.091362,
    21.09200478,
    21.09267616,
```

```
    104.0042877,
    103.6608887,
    104.0093689,
    103.9963226,
    104.005722,
    104.0260315,
    104.020462,
    104.0037918,
    104.0249329,
    104.0223846,
    103.9259949,
    104.0123291,
    104.0151291,
    103.7541656,
    104.0238266,
    103.8874207,
    104.0103149,
    104.0194855,
    103.8282471,
    104.0186157,
    104.0200272,
    104.0175095,
    104.0224991,
    104.0059433,
    104.0204239,
    104.0163651,
    104.0129852,
    104.0053864,
    104.008461,
    104.0100403,
    104.0093155,
    104.0151291,
    104.009613,
    104.0077057,
    104.0057602,
    104.0035553,
    104.000145,
    104.0070419,
    103.9960175,
    104.0026093,
    104.0076141,
    104.0126724,
    103.9877014,
    104.0131302,
    104.011734,
    104.0048676,
    80.06158447,
    103.987175,
```

```
    21.09349632,
    21.09443092,
    21.09471321,
    21.09518623,
    21.09583092,
    21.09710693,
    21.09830666,
    21.09830856,
    21.09853935,
    21.09897232,
    21.10046959,
    21.10069847,
    21.10279274,
    21.1040554,
    21.10406303,
    21.10447884,
    21.10457802,
    21.10482979,
    21.10523605,
    21.10632324,
    21.10739326,
    21.10821533,
    21.10977936,
    21.11573029,
    21.11862183,
    21.12489128,
    21.12633514,
    21.12875748,
    21.12911606,
    21.13218689,
    21.14027405,
    21.14509201,
    21.14734077,
    21.1509819,
    21.15248489,
    21.15282631,
    21.15327835,
    21.15489578,
    21.15504265,
    21.15717125,
    21.15751648,
    21.15879822,
    21.16286659,
    21.16297531,
    21.16464806,
    21.16486359,
    21.16488457,
    21.16497231,
```

```
    103.985466,
    103.9895401,
    103.9834366,
    103.9838943,
    103.9847183,
    103.9933701,
    103.9929581,
    104.0065994,
    103.9857254,
    103.9970932,
    103.9876022,
    104.0078583,
    103.9985428,
    103.9933395,
    80.03785706,
    104.0032196,
    103.9949188,
    104.0057068,
    103.9967194,
    103.9964142,
    104.0035934,
    104.0037231,
    103.9881439,
    103.9905777,
    80.01541138,
    0.633025885,
    1.010762334,
    103.9920578,
    103.9875717,
    103.9842377,
    103.9802551,
    0.72463268,
    103.9863358,
    103.9772415,
    0.737806439,
    0.666772783,
    103.9753189,
    0.71250397,
    104.0016174,
    0.649995625,
    0.56504482,
    103.9825058,
    103.9962006,
    0.548948944,
    0.417595267,
    0.516443133,
    0.572074711,
    0.606859803,
```

```
    21.16744614,
    21.16775131,
    21.16832161,
    21.17133331,
    21.17160416,
    21.17471123,
    21.17508316,
    21.17716217,
    21.18076134,
    21.18158722,
    21.1832943,
    21.18564606,
    21.18933678,
    21.1901226,
    21.1904335,
    21.19178009,
    21.19297409,
    21.19408035,
    21.19504929,
    21.19533539,
    21.195961,
    21.19838715,
    21.20091629,
    21.20517159,
    21.20532036,
    21.21362305,
    21.21375275,
    21.21441269,
    21.2152195,
    21.21564102,
    21.21688461,
    21.21689606,
    21.21711349,
    21.21739388,
    21.21798706,
    21.2188015,
    21.21956825,
    21.22012711,
    21.22019577,
    21.22077751,
    21.22094154,
    21.2211132,
    21.22214508,
    21.22216606,
    21.22240067,
    21.22309113,
    21.22329712,
    21.22468185,
```

```
    103.985672,
    103.9974136,
    103.9943771,
    103.9875717,
    79.96071625,
    103.9767075,
    0.548192263,
    0.639898777,
    103.975647,
    103.9835663,
    0.625411451,
    103.9785995

]
```

```
    21.22485924,
    21.22523689,
    21.22573853,
    21.22593498,
    21.22731781,
    21.2289772,
    21.22955132,
    21.22991371,
    21.23052597,
    21.23059082,
    21.23218918,
    21.23358154

]
```

Serta terdapat pula *dependency* untuk keperluan fitur lain, yaitu:

InputGauss.py

```
inMatrix = [
    [3, -0.1, -0.2],
    [0.1, 7, -0.3],
    [0.3, -0.2, 10],
]

inVector = [
    7.85, -19.3, 71.4
]
```

C. ANALISIS

I. Hasil Awal Simulasi

(hasil simulasi sementara)

```
Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[21.142175482336917, -0.0019743529191370273, -3.514245380498358e-05, 5.282738257947516e-07]                    with 31591 iterations.

Translated into Power Regressor in a function of:
5.282738257947516e-7*x**3 - 3.514245380498358e-5*x**2 - 0.0019743529191370273*x + 21.142175482336917

Analysis of Root:
Newton Raphson resulted in:
6975.42627261012
with 2 iterations.
Secant Method  resulted in:
Failed
with Failed iterations.

[Evaluation completed]
```

Pada proses regresi linear menggunakan Power Regression berderajat 3 dengan bantuan Gauss-Seidel, ditemukan fungsi regresinya dengan total 31591 iterasi yang terjadi. Kemudian, fungsi tersebut langsung dimasukkan ke dalam kedua metode Root-Finding, yaitu Newton Raphson dan Secant Method. Kedua metode ini tidak semuanya berjalan dengan mulus. Pada metode Newton-Raphson, iterasi berhasil dilakukan hanya 2 kali saja dengan proses yang berhenti karena limit toleransi tercapai. Sementara Secant-Method tidak berhasil menemukan akar sehingga terjadi infinite recursion yang kemudian di-handle dengan error exception.

```
You're choosing: Exported Input
You're choosing: Data Set -> Power Regression --[GS]--> Function --[NR, SM]--> Root

Start to Evaluate Gauss Seidel Method
Terminated by tolerated error value
Result: [21.142175482336917, -0.0019743529191370273, -3.514245380498358e-05, 5.282738257947516e-07]
Error : [9.683989982183601e-08%, 0.0009999445866974675%, 0.0012920188355176273%, 0.0004849837484538525%]

Start to evaluate Newton Raphson
Stopped by the error tolerance limit
Newton Raphson Stopped! Last Value: 4657.56702530131

Start to evaluate Secant Method
Exception Error
```

II. Variasi Ukuran Matriks

(karena terdapat beberapa hasil yang tidak relevan dalam percobaan, maka percobaan variasi ukuran matriks belum dapat dilaksanakan)

III. Analisis Hasil

(analisis sementara)

Fungsi yang ditemukan secara garis besar memiliki koefisien pangkat tertinggi yang bernilai positif, sehingga hal ini menunjukkan bahwa panas magnet motor elektrik akan selalu meningkat seiring dengan meningkatnya torsi. Tentu hal ini cukup logis untuk menyatakan bahwa tidak mungkin ada titik di mana torsi meningkat, tetapi panas magnet motor elektrik justru menurun. Dalam hal ini, metode Root-Finding akhirnya menemukan titik di mana suhu magnet motor mulai meningkat, yaitu pada saat torsi bernilai sekitar 4657 Nm.

D. KESIMPULAN

(menimbang perlunya analisis kasus yang lebih mendalam sekaligus pengembangan kode yang menyesuaikan kebutuhan soal, maka kesimpulan belum dapat diformulasikan)