LAPORAN TUGAS BESAR METODE NUMERIS: OPTIMASI DAN ANALISIS HUBUNGAN TEMPERATUR UDARA TERHADAP TITIK EMBUN KOTA İZMIR, TURKI PADA TAHUN 2017

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 dampak perubahan temperatur udara terhadap perubahan titik embun di kota İzmir, Turki pada tahun 2017. Data yang digunakan pada laporan ini diambil dari Kaggle dengan link berikut ini: https://www.kaggle.com/datasets/ibrahimkiziloklu/solar-radiation-dataset/data. Data tersebut sebenarnya menampilkan berbagai faktor pendukung perubahan titik embun (*dew point*) selain temperatur, tetapi temperatur dinilai sebagai kontributor paling unggul yang kemudian dijadikan parameter dalam analisis laporan ini. Adapun permasalahan yang diulas pada laporan ini adalah terkait pemahaman hubungan temperatur udara dengan perubahan titik embun pada kota tersebut. Kemudian, laporan ini juga membahas bagaimana bentuk optimasi yang diperlakukan kepada hubungan yang ditemukan untuk menentukan titik maksimum perubahan titik embun.

Kode dari program ini dapat diakses secara publik melalui link GitHub berikut ini: https://github.com/iZcy/MetNum-PraUTS. Adapun pada link tersebut terdapat beberapa folder dan file tambahan seperti variasi bentuk program dan data set yang digunakan. Hal ini dilakukan untuk mempermudah pembacaan hasil analisis pada data.

II. Tujuan Simulasi Numeris

Tujuan dari perlakuan simulasi numeris ini di antaranya adalah,

a. Mengetahui hubungan antara nilai temperatur udara dan titik embun di kota İzmir, Turki pada tahun 2017.
b. Mengetahui saat ketika titik embun maksimum diperoleh sebagai dampak dari perubahan nilai temperatur udara.

III. Kekangan Simulasi

Simulasi ini sebenarnya tidak memerlukan adanya perlakuan kekangan karena apabila dilihat dari persebaran data, nilai-nilai suhu yang berlaku tidaklah melebihi batas kemampuan suhu yang dapat dikompromi oleh manusia. Akan tetapi, sebagai uji coba tambahan, maka ditetapkan kekangan minimum suhu sebesar $10°C$ dan suhu maksimum $27°C$ sebagai interval suhu yang masih terhitung normal di daerah iklim tropis. Meskipun Turki bukan merupakan daerah beriklim tropis, interpretasi ini dapat menunjukkan bagaimana perbandingan keadaan udaranya terhadap negara tropis seperti Indonesia.

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 data set menjadi sebuah persamaan linear. Kemudian, persamaan linear ini diturunkan lalu diteruskan kepada Root-Finding Methods, yaitu Newton Raphson dan Secant Method untuk ditemukan akar-akarnya yang akan menandakan titik optimum dari fungsi yang diregresi sebelumnya. Terakhir, nilai-nilai ini dievaluasi kembali terhadap fungsi regresi untuk menunjukkan nilai maksimum / minimum.

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

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

II. Estimasi Awal

Estimasi yang digunakan untuk menyelesaikan masalah ini dapat diterapkan untuk penyelesaian Linear Equation dan penyelesaian Root Finding. Adapun estimasi yang dilakukan adalah sebagai berikut:

1. Linear Equation

   Estimasi ini diterapkan untuk metode Gauss-Seidel dan esitmasi dibiarkan *default*, yaitu vektor nol. Alasan vektor nol digunakan adalah karena bentuk data yang terlalu luas dan sulit untuk ditebak koefisien polinomnya.
2. Root Finding

   Estimasi ini diterapkan untuk metode Newton Raphson dan Secant Method. Adapun sesuai dengan kekangan yang telah ditentukan, maka estimasi diambil secara *random* di antara kedua interval yang ditentukan. Pada laporan ini, digunakan nilai $14°C$ dan $16°C$ sebagai kedua titik estimasi untuk kedua metode Root Finding.

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 eror, jumlah nilai tertoleransi minimum yang diizinkan, dan limit maksimal iterasi. Terakhir, pada kedua metode Root-Finding yang digunakan, hanya diterapkan nilai toleransi eror dan limit maksimal iterasi yang sama nilainya seperti pada limitasi metode Gauss-Seidel.

Pada program yang dibuat, kriteria ini dapat diisi dengan cara mengisi manual kepada program atau dengan melakukan konfigurasi langsung dalam kodenya. Oleh karena itu, program ini dibuat dengan memungkinkan pengguna untuk melakukan input manual ke dalam program apabila data yang perlu dimasukkan sedikit, serta melakukan input langsung ke dalam kode apabila data yang perlu dimasukkan tergolong banyak. Adapun konfigurasi yang langsung dimasukkan ke dalam kode adalah sebagai berikut.

```
# Setting Up Configurations
# Method Setting
regressPower = 4      # Regressor Power + 1 ex. for 1 degree of x, do 2
gseidMinErrPass = 0   # Default<=0 adjust to the vector size
# 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 = 14
inputVal = 16
inputGuess = []  # n = Regressor power: otherwise, return error
# [alternative: leave as an empty array to automatically make zero vector
guess]

# Round and Tolerance
inputRound = 50
errRound = 50
errTolerance = 1e-100
iterLimit = 1000

# Optimization
intervalMin = 10
intervalMax = 24

# View Toggle
viewProcess = False
```

IV. Pseudocode

Pada program yang dibuat, sebenarnya terdapat beberapa fungsi tambahan yang dibuat untuk mempermudah *interface* dengan pengguna dan menghubungkan fungsi-fungsi utama tergantung kepada opsi yang dipilih oleh pengguna. Oleh karena itu, *pseudocode* yang akan ditampilkan pada laporan ini hanyalah *pseudocode* untuk keempat metode numeris yang digunakan.

1. Power Regressor

Meskipun nama dari fungsi ini adalah Power Regressor, sebenarnya fungsi ini adalah fungsi dependen yang tugasnya hanyalah untuk menghasilkan matriks yang memiliki solusi berupa koefisien dari masing-masing pangkat polinomial yang dibutuhkan untuk melakukan Power Regression (atau lebih tepatnya Polynomial Regression). Adapun rumus matriks yang dimaksud adalah rumus matriks berikut.

$$\begin{bmatrix} n & \sum x_i & \dots & \sum x_i^m \\ \sum x_i & \sum x_i^2 & \dots & \sum x_i^{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum x_i^m & \sum x_i^{m+1} & \dots & \sum x_i^{2m} \end{bmatrix} \begin{bmatrix} koef.x \\ koef.x^2 \\ \vdots \\ koef.x^m \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^m y_i \end{bmatrix}$$

Sebagai implementasi dari pembentukan matriks tersebut, maka *pseudocode* yang digunakan adalah seperti berikut.

```
## Code for PowerRegressor

    #Power Regressor, Array Formulator
    Function of PowerRegressor accepts:
    x_val (default []),
    y_val (default []),
    power (default 1)

    # Exception Handler
    If length Of x_val Is Not Equal To y_val
        Print "Invalid input! Cannot regress different size of x_val and
y_val!"
        STOP THE PROGRAM

    # Evaluate Matrix
    # Take the n
    Set n_val To length Of x_val  # Bascially the same as length of y_val
    Set matrix As empty array

    # Iterate through matrix
    For In Range Of power use Index_i
        Set vector As empty array

        # Iterate through vector
        For In Range Of power use Index_j
```

```
            # Only append if it's not the very first element
            If Index_i Equal To 0 And Index_j Equal To 0
                Append n_val To vector # It is so that the first element of
the matrix is n (according to the polynomial regression matrix formula)
                SKIP LOOP

            # Iterate through sum
            Set sum As 0
            For In Range Of length of x_val use Index_k
                # Sigma of x_val^(row+col)
                Add x_val[k] Power To (i + j) To sum

            Append sum To vector

        Append vector To matrix

    # Evaluate Res_Vector
    Set res_vector As empty array

    # Iterate through y
    For In Range power use Index_i
        Set sum As 0
        # Sum each power
        For In Range Of length of y_val use Index_j
            # Component y & x
            Set y_comp As y_val[j]
            Set x_comp As x_val[j] Power to i

            # Sum
            Add y_comp Times x_comp To sum

        # Append vector
        Append res_vector To Sum

    return [matrix, res_vector]
```

2. Gauss Seidel

Formulasi algoritma Gauss-Seidel yang digunakan adalah sebagai berikut.

$$koef.x_{i+1} = \frac{\sum y_i \; - \; konst.x^2 * koef.x^2 \; - \; \ldots \; - \; konst.x^m * koef.x^m}{konst.x}$$

$$koef.x^2{}_{i+1} = \frac{\sum x_i y_i \; - \; konst.x_{i+1} * koef.x_{i+1} \; - \; \ldots \; - \; konst.x^m * koef.x^m}{konst.x^2}$$

$$\vdots$$

$$koef.x^m{}_{i+1} = \frac{\sum {x_i}^m y_i \; - \; konst.x_{i+1} * koef.x_{i+1} \; - \; konst.x^2{}_{i+1} * koef.x^2{}_{i+1} \; - \; \ldots \; - \; konst.x^{m-1} * koef.x^{m-1}}{konst.x^m}$$

Formulasi tersebut disusun dalam bentuk program dengan *pseudocode* seperti berikut.

```
## Code for GaussSeidel

    #Gauss-Seidel

    Function of GaussSeidel accepts:
    roundV,
    errTol,
    totalTerm,
    b_val,
    matrix,
    guess (default 0),
    limit (default 100),
    view (default True)

    # Exception Handler
    If guess Is Not Equal To 0 And length of guess Is Not Equal To b_val
        print "Inequal length of 'guess-vector' and 'y-vector' Terminating the
whole process!"
        return [0, 0] # Defined as False / Failing condition

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

    # Init Values
    Set iteration As 0
    Set arrOut As empty array
    Set errEnd As empty array

    Set errX As empty array
    Set x_valProcess As empty array

    print b_val + "\n" + matrix

    # Guess-Handler
    If guess Equal To 0 # If it is not set, set all to zero
        For In Range length of b_val use Index_i
            Append 0 to errX
```

```
        Copy Elements errX To x_valProcess

    Else # If it is set, copy it to process and error comparison
        Set x_valProcess As guess
        Copy Elements guess To errX

    # Loop the Gauss-Seidel
    For In Range of limit use Index_k
        # Terminate Message
        Set termMsg As ""

        If view Is True
            print "Loop :" + (k+1) + "\nBefore: "

            # Current value Display
            Display x_valProcess Rounded to roundV

        # Single Gauss-Seidel Process
        Copy Elements x_valProcess To prev_x_valProcess

        For In Range length Of matrix use Index_j
            Set val As 0
            # Adding value to the b-value
            Add b_val[j] To val

            For In Range length of matrix[j]-1 use Index_i
                Set index as (j + 1 + i) Mod length of matrix[j]

                # Subtracting the value to x vars with response to the DYNAMIC
x-input
                Subtract val To matrix[j][index] * x_valProcess[index]

            # Dividing the value to the analyzed x-output
            Divide val To matrix[j][j]

            # Move value to the storage
            Set x_valProcess[j] As val

        # Error Calculation
        Set triggerTerminate As False

        # Error Tolerance Count
        Set toleratedError As 0
        Set zeroError As 0

        If view Is True
            print "Current: "
```

```
            # Current value Display
            Display x_valProcess Rounded to roundV

        For In Range length of errX use Index_i
            Set errX[i] As 0

            TRY
              Set errX[i] As Absolute of ((x_valProcess[i] -
prev_x_valProcess[i]) / x_valProcess[i])*100
            EXCEPT
              Set errX[i] As 0

            # Terminate if the minimum error achieved
            If Absolute of errX[i] < errTol
                Add 1 to toleratedError
                If triggerTerminate is False And toleratedError Equal to
totalTerm
                    Set termMsg As "Terminated by tolerated error value"
                    Set triggerTerminate As True

            If errX[i] Equal to 0
                Add 1 to zeroError
                If triggerTerminate is False and zeroError Equal to totalTerm
                    Set termMsg As "Terminated by zero error achieved"
                    Set triggerTerminate As True

        If view is True
            print "Error: "
            Display errX Rounded to roundV

        # Stop if there exist tolerable error
        If triggerTerminate is True
            # Save the Result
            Set iteration As k+1
            Set arrOut As x_valProcess
            Set errEnd As errX
            BREAK THE LOOP

        # Save if Limit
        If k Equal to (limit - 1)
            Set termMsg As "Terminated by Limit of iteration"
            Set iteration As k+1
            Set arrOut As x_valProcess
            Set errEnd As errX

    print termMsg
    print "Result: "
    Display x_valProcess Rounded to roundV
```

```
    print "Error : "
    Display errEnd Rounded to roundV

    return [arrOut, iteration]
```

3. Newton Raphson
   Formulasi algoritma Newton Raphson yang digunakan adalah sebagai berikut.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Formulasi tersebut disusun dalam bentuk program dengan *pseudocode* seperti berikut.

```
## Code for NewtonRaphson

    #NewtonRaphson
    Function of NewtonRaphson accepts:
    f,
    f_dif,
    val,
    roundV,
    errRound,
    tolerance (default 0.00001),
    limit (default 1000),
    view (default True),
    intervalMin (default -infinity),
    intervalMax (default infinity)

    Set iter As 0
    Set errPrev As 0
    currentVal = val

    For In Range Of limit use Index_it
        Set iter As it + 1

        # Intializing Process
        If Index_it Equal to 0
            print "Start to evaluate Newton Raphson"

        # Newton Raphson Formula + Error EcurrentValuation
        Set currentValue As currentVal - f(currentVal) / f_dif(currentVal)
        Set error As (currentValue - currentVal)/currentValue

        If view Equal to True
            print "Iterate: " + (it+1) + "\tPrevious: " + currentVal +
"\tCurrent: " + currentValue + "\tError: " + Round (error*100) to errRound +
"%"

        # Updating Process
```

```
        Set terminate As False

        Set errorStop As False # To allow linear functions be able to
determined

        TRY
          Absolute of (error - errPrev) Less Than tolerance
        EXCEPT
          Do Nothing

        # Simulation Terminator: Interval exceeding --- Exceeds the
restriction
        If (intervalMin Is Not Equal To -infinity and intervalMax Is Not Equal
To infinity) And (currentValue Less Than intervalMin Or currentValue Greater
Than intervalMax)):
            print "Stopped by the restricted interval exceeded"

            If (currentValue Less Than intervalMin)
                Set currentValue As intervalMin
            Elif (currentValue Greater Than intervalMax)
                Set currentValue As intervalMax

            return [currentValue, iter]

        # Simulation Terminator: Found --- 0 Error a.k.a currentValue is found
        If error Equal To 0
            print "Stopped by the exact currentValue found"
            Set terminate As True

        # Simulation Terminator: Error Tolerance --- Error difference is too
small to continue
        Elif (it Is Not Equal To 0 And errorStop Is True)
            print "Stopped by the error tolerance limit"
            Set terminate As True

        # Simulation Terminator: Almost exact --- Change of currentValue is
too small to continue
        Elif Round currentVal To roundV Equal To Round currentValue To roundV
            print "Stopped by the tolerated round currentValue"
            Set terminate As True

        # Simulation Terminator: Found!
        Elif f(currentValue) Equal To 0.0
            print "Solution found!"
            Set terminate As True

        # Continue Update
        Set currentVal As currentValue
```

```
        # Terminator
        If terminate Is True
            print "Newton Raphson Stopped! Last currentValue:" + currentValue
            return [currentVal, iter]

    # Simulation Terminator: Limit of Loop
    print "Iteration Limit! The function is either divergent or requires more
iteration!"
    return [currentVal, iter]
```

4. Secant Method
   Formulasi algoritma Secant Method yang digunakan adalah sebagai berikut.

$$x_{i+1} = x_i - f(x_i) * \frac{(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

Formulasi tersebut disusun dalam bentuk program dengan *pseudocode* seperti berikut.

```
## Code for SecantMethod

##SecantMethod


    Function of SecantMethod accepts:
    f,
    valCurr,
    valBef,
    roundV,
    errRound,
    tolerance (default 0.00001),
    limit (default 1000),
    view (default True),
    intervalMin (default -infinity),
    intervalMax (default infinity):

    Set iter As 0
    Set errPrev As 0

    Set currentVal As valCurr
    Set current2ndVal As valBef

    For In Range limit use index_it
        Set iter As it + 1

        # Intializing Process
        If it Equal To 0
            print "Start to evaluate Secant Method"
```

```
        # Secant Method Formula + Error Evaluation
        Set valUpd As currentVal - (
            f(currentVal) * (
                current2ndVal - currentVal
            )
        ) / (
            f(current2ndVal) - f(currentVal)
        )
        Set error As (valUpd - currentVal) / valUpd

        If view Equal To True
            print "Iterate: " + (it+1) + "\tPrevious: " + currentVal +
"\tCurrent: " + valUpd + "\tError: " + Round error*100 To errRound + "%"

        # Updating Process
        Set terminate As False

        # Simulation Terminator: Interval exceeding --- Exceeds the
restriction
        If (intervalMin Is Not Equal To -infinity) And (intervalMax Is Not
Equal To infinity) And (valUpd Less Than intervalMin Or valUpd Greater Than
intervalMax)
            print "Stopped by the restricted interval exceeded"
            If valUpd Less Than intervalMin
                Set valUpd As intervalMin
            Elif valUpd Less Than intervalMax
                Set valUpd As intervalMax

            return [valUpd, iter]

        # Simulation Terminator: Found --- 0 Error a.k.a value is found
        If error Equal To 0
            print "Stopped by the exact value found"
            Set terminate As True

        # Simulation Terminator: Error Tolerance --- Error difference is too
small to continue
        Elif it Is Not Equal To 0
            TRY
                If Absolute of (error - errPrev) Less Than tolerance
                    print "Stopped by the error tolerance limit"
                    Set terminate As True
            EXCEPT
                0

        # Simulation Terminator: Almost exact --- Change of value is too small
to continue
        Elif Round currentVal To roundV Equal To Round valUpd To roundV
```

```
            print "Stopped by the tolerated round value"
            Set terminate As True

        # Simulation Terminator: Found!
        Elif f(valUpd) As 0.0
            print "Solution found!"
            Set terminate As True

        # Continue Update
        Set current2ndVal As currentVal
        Set currentVal As valUpd

        # Terminator
        If terminate Is True
            print "Secant Method Stopped!  Last Value:" + valUpd
            return [Round valUpd To roundV, iter]

    # Simulation Terminator: Limit of Loop
    print "Iteration Limit! The function is either divergent or requires more
iteration!"
    return [Round currentVal To roundV, iter]
```

V. Code

Seperti yang telah dijelaskan sebelumnya, kode yang digunakan telah dibentuk sedemikian rupa untuk memiliki kemampuan diisi secara *encoded* dalam kode langsung atau dengan melakukan *input* manual. Namun, dalam laporan ini, fitur yang digunakan adalah fitur *encoded* dalam kode langsung karena banyaknya data yang di-*input* adalah 9000 pasang yang tidak mungkin di-*input* manual satu per satu.

Kode ini pada dasarnya terbagi menjadi beberapa part, yaitu:

1. Numerical Methods Functions: PowerRegressor, GaussSeidel, NewtonRaphson, dan SecantMethod
2. Input Handler Functions: Config, InputX, InputY, InputGauss, dan InputFunction
3. Middleware Functions: Processors, ExtraFunction, dan DataValidity
4. Display Functions: Interface dan Output

Modul-modul ini berjalan dengan prekursor berupa main *file*. Akan tetapi, sesuai dengan permintaan, keseluruhan *package* terpisah tersebut dijadikan satu *file* dengan nama compactMain.py (satu *file* yang sudah terkompilasi). Sangat disarankan untuk melihat kode langsung melalui link GitHub ini: https://github.com/iZcy/MetNum-PraUTS. Salah satu folder yang ada dalam *repository* itu adalah Modulated yang berisi *file packaging* yang lebih rapi dan lebih mudah untuk dimengerti. Adapun prekursor pada folder tersebut adalah main.py. Akan tetapi, untuk memenuhi permintaan, maka kode yang disusun dalam satu program juga ditampilkan pada laporan ini.

```
# Setting Up Configurations
# Method Setting
regressPower = 4      # Regressor Power + 1 ex. for 1 degree of x, do 2
gseidMinErrPass = 0   # Default<=0 adjust to the vector size
# 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 = 41
inputVal = 45
inputGuess = []  # n = Regressor power: otherwise, return error
# [alternative: leave as an empty array to automatically make zero vector
guess]

# Round and Tolerance
inputRound = 50
errRound = 50
errTolerance = 1e-100
iterLimit = 1000

# Optimization
intervalMin = float('-inf')
intervalMax = float('inf')

# View Toggle
viewProcess = False
#
#
#
#
#
#
#
#
#
#
#
# IMPORT & Symbols
from sympy import *
import os
import copy

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

```
#
#
#
#
#
# INPUT
x_vector = [
    0,
    1,
    1.7071067811865
]

y_vector = [
    1,
    -1,
    0
]

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

inVector = [
    7.85, -19.3, 71.4
]

funct = "-2*x**2 + 3*x + 1"

try:
  funct = sympify(funct)
except:
  print("Invalid Function Input!")
  exit()

# Check Matrix Validity
vctLen = len(inMatrix[0])
for i in range(len(inMatrix) - 1):
  currVctLen = len(inMatrix[i + 1])

  # print(f"Compare {vctLen} vs {currVctLen}")
  if (vctLen != currVctLen):
    print("Inequal inMatrix Size!")
    exit()

  vctLen = currVctLen
```

```
# Check Vector Validity
if (vctLen != len(inVector)):
  print("Incompatible inVector Size!")
  exit()

#
#
#
#
#
#
#
#
#
#
# NUMERICAL METHODS
# 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 = []

    # print(b_val, "\n", matrix)

    # 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] = 0
            try:
              errX[i] = abs((x_valProcess[i] - prev_x_valProcess[i]) /
                            x_valProcess[i])*100
            except:
              errX[i] = 0

            # 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]

# NewtonRaphson
def NewtonRaphson(f, f_dif, val, roundV, errRound, tolerance=0.00001,
limit=1000, view=True, intervalMin=float('-inf'), intervalMax=float('inf')):
    iter = 0
    errPrev = 0
    currentVal = val

    for it in range(limit):
        iter = it + 1
        # Intializing Process
        if (it == 0):
            print("Start to evaluate Newton Raphson")

        # Newton Raphson Formula + Error EcurrentValuation
        currentValue = currentVal - evToX(f, currentVal) / evToX(f_dif,
currentVal)
        error = (currentValue - currentVal)/currentValue

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

        # Updating Process
        terminate = False

        errorStop = False # To allow linear functions be able to determined
        try:
          abs(error - errPrev) < tolerance
        except:
          0

        # Simulation Terminator: Interval exceeding --- Exceeds the
restriction
        if ((intervalMin != float('-inf') and intervalMax != float('inf')) and
(currentValue < intervalMin or currentValue > intervalMax)):
            print("Stopped by the restricted interval exceeded")
            if (currentValue < intervalMin):
                currentValue = intervalMin
            elif (currentValue > intervalMax):
                currentValue = intervalMax

            return [currentValue, iter]

        # Simulation Terminator: Found --- 0 Error a.k.a currentValue is found
```

```
        if (error == 0):
            print("Stopped by the exact currentValue found")
            terminate = True

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

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

        # Simulation Terminator: Found!
        elif (evToX(f, currentValue) == 0.0):
            print("Solution found!")
            terminate = True

        # Continue Update
        currentVal = currentValue

        # Terminator
        if terminate:
            print("Newton Raphson Stopped! Last currentValue:", currentValue)
            return [currentVal, iter]

    # Simulation Terminator: Limit of Loop
    print("Iteration Limit! The function is either divergent or requires more
iteration!")
    return [currentVal, iter]

# 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!")
        exit()

    # 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]
```

```
# Secant Method


def SecantMethod(f, valCurr, valBef, roundV, errRound, tolerance=0.00001,
limit=1000, view=True, intervalMin=float('-inf'), intervalMax=float('inf')):
    iter = 0
    errPrev = 0

    currentVal = valCurr
    current2ndVal = valBef

    for it in range(limit):
        iter = it + 1

        # Intializing Process
        if (it == 0):
            print("Start to evaluate Secant Method")

        # Secant Method Formula + Error Evaluation
        valUpd = currentVal - (
            evToX(f, currentVal) * (
                current2ndVal - currentVal
            )
        ) / (
            evToX(f, current2ndVal) - evToX(f, currentVal)
        )
        error = (valUpd - currentVal)/valUpd

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

        # Updating Process
        terminate = False

        # Simulation Terminator: Interval exceeding --- Exceeds the
restriction
        if ((intervalMin != float('-inf') and intervalMax != float('inf')) and
(valUpd < intervalMin or valUpd > intervalMax)):
            print("Stopped by the restricted interval exceeded")
            if (valUpd < intervalMin):
                valUpd = intervalMin
            elif (valUpd > intervalMax):
                valUpd = intervalMax

            return [valUpd, iter]
```

```
        # 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):
            try:
                if (abs(error - errPrev) < tolerance):
                    print("Stopped by the error tolerance limit")
                    terminate = True
            except:
                0

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

        # Simulation Terminator: Found!
        elif (evToX(f, valUpd) == 0.0):
            print("Solution found!")
            terminate = True

        # Continue Update
        current2ndVal = currentVal
        currentVal = valUpd

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

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

#
#
#
#
#
#
#
#
```

```
#
#
# MIDDLEWARE --- [Data Validation]
def checkValidity():
    if (len(x_vector) != len(y_vector)):
        print("Data size mismatch!")
        exit()


def checkConfig(resLen):
    # print(f"Len: {resLen} with gseiderr: {gseidMinErrPass} and arrLenGuess:
{len(inputGuess)} cond1: {gseidMinErrPass > resLen} and cond2:
{len(inputGuess) > resLen}")
    if ((gseidMinErrPass > resLen) or (len(inputGuess) > resLen)):

        print("Config size conflicts! Please rematch your config!")
        exit()

# MIDDLEWARE --- [Extra Function]
# 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="")

def printMatrix(mtr, roundV, postfix=""):
    for i in range(len(mtr)):
        printArray(mtr[i], roundV, postfix)
        print()

# 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

def promptError(errMsg, loopMsg):
    os.system('cls')
    print(errMsg)
    if (loopMsg != ""):
        print(loopMsg)

def mustNumber(msg, errMsg="Invalid input! Please try again!",
dataType="float", loopMsg="", accept=[]):
    value = ""
    while(True):
        value = input(msg)
        if value in accept:
            break

        try:
            if (dataType == "float"):
                value = float(value)
                break
            elif (dataType == "int" or dataType == "unsigned"):
                value = int(value)
                if (dataType == "int"):
                    break
                elif (dataType == "unsigned"):
                    if (value <= 0):
                        promptError(errMsg, loopMsg)
                    else:
                        break
            continue
        except:
            promptError(errMsg, loopMsg)

    return value
```

```
# MIDDLEWARE --- [Processors]
# Execute Function
def PowRegProcess(inputVectX=[], inputVectY=[], power=regressPower):
    os.system('cls')
    # Calling Methods for Power Regressor
    if (inputVectX != [] and inputVectY != []):
        matrixPR, vectorPR = PowerRegressor(
            x_val=(inputVectX), y_val=(inputVectY), power=(power))
    else:
        matrixPR, vectorPR = PowerRegressor(
            x_val=(x_vector), y_val=(y_vector), power=(power))

    # print(matrixPR)
    # print(vectorPR)

    return [matrixPR, vectorPR]


def GauSedProcess(matrixPR, vectorPR, convertFunc=True, guess=(inputGuess if
inputGuess else 0), errTol=(errTolerance)):
    os.system('cls')
    # Calling Methods for Linear Algebra Solving

    # Default minError = as much as vector elements
    minErr = 0
    if gseidMinErrPass > 0:
        minErr = gseidMinErrPass
    else:
        minErr = len(vectorPR)

    gaussSeidRes, gaussSeidIter = GaussSeidel(roundV=(inputRound),
errTol=(errTol), totalTerm=(
        minErr), b_val=(vectorPR), guess=(guess), matrix=(matrixPR),
limit=(iterLimit), view=(viewProcess))

    if gaussSeidIter == 0:
        return
    print()

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

    return [fixedFunction, gaussSeidRes, gaussSeidIter]
```

```
def SrRootProcess(fixedFunction, intervalMin=(intervalMin),
intervalMax=(intervalMax), guessOne=(inputVal), guessTwo=(inputBefVal),
errTol=(errTolerance)):
    os.system('cls')
    # Calling Methods Root-Finding
    newtRaphRes, newtRaphIter = NewtonRaphson(f=(fixedFunction),
f_dif=(diff(fixedFunction, x)), val=(guessOne), roundV=(
        inputRound), errRound=(errRound), tolerance=(errTol),
limit=(iterLimit), view=(viewProcess), intervalMin=(intervalMin),
intervalMax=(intervalMax))
    print()
    secnMthdRes, secnMthdIter = SecantMethod(valCurr=(guessOne),
valBef=(guessTwo), roundV=(
        inputRound), errRound=(errRound), tolerance=(errTol),
f=(fixedFunction), limit=(iterLimit), view=(viewProcess),
intervalMin=(intervalMin), intervalMax=(intervalMax))
    print()

    return [newtRaphRes, newtRaphIter, secnMthdRes, secnMthdIter]

#
#
#
#
#
#
#
#
#
#
# DISPLAY --- [Output]
def analysis(gaussSeidRes, gaussSeidIter, fixedFunction, realFunction,
newtRaphRes, newtRaphIter, secnMthdRes, secnMthdIter, isRootFinding,
isLinEqSlving, isEqRegressor, manualInput=False, x_data=[], y_data=[],
isOptimization = False):
    # Setup Import
    analyzeX = x_vector
    analyzeY = y_vector

    if manualInput:
        analyzeX = x_data
        analyzeY = y_data

    # Making Up Conclusions:
    if (isEqRegressor):
        print("[Evaluation finished with the result]\n")
        # print(f"Analyzing the dataset of:\n",
        #       f"x = {analyzeX}\n",
```

```
        #       f"y = {analyzeY}\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{fixedFunction}", end="\n\n")

    if (isRootFinding):
        print("Analysis of Root:")

        if (isOptimization):
            print("\n(Optimization Mode)\n")

        print(
            f"Newton Raphson resulted in:\n{newtRaphRes}\nwith {newtRaphIter}
iterations.")

        if (isOptimization):
            print("Max/min Value:", evToX(realFunction, newtRaphRes))

        print(
            f"Secant Method  resulted in:\n{secnMthdRes}\nwith {secnMthdIter}
iterations.")

        if (isOptimization):
            print("Max/min Value:", evToX(realFunction, secnMthdRes))
        print("\n[Evaluation completed]")


def execute(isRootFinding, isLinEqSlving, isEqRegressor, manualInput,
isOptimization = False):
    # print(f"{isRootFinding}, {isLinEqSlving}, {isEqRegressor},
{inputManual}")
    convertToFunc = False
    inputManual = list(manualInput)[0]

    # Setting Error Tolerance
    errTol = 0
    if (inputManual):
        errTol = mustNumber("Please input error tolerance: ",
dataType=("float"))
    os.system('cls')

    # Calling PowerReg
```

```
    matrixPR = []
    vectorPR = []
    if (isEqRegressor and not inputManual):
        matrix, vector = PowRegProcess()
        matrixPR = copy.deepcopy(matrix)
        vectorPR = copy.deepcopy(vector)
        convertToFunc = True

    # Function Definition
    fixedFunction = funct
    realFunction  = funct

    while (True and inputManual and isRootFinding and not isLinEqSlving):
        tempFunct = input("Please input your function: ")

        passed = False
        try:
            tempFunct = sympify(tempFunct)
            round(evToX(tempFunct, 1), 1)
            passed = True
        except:
            os.system('cls')
            print("Your input is not a valid function!")

        if (passed):
            fixedFunction = tempFunct
            realFunction  = tempFunct
            break

    # Calling GaussSed

    gaussSeidRes = []
    gaussSeidIter = 0

    if (inputManual):
        if (isLinEqSlving):
            repeatX = mustNumber(
                "Please input your data row: ", dataType="int")
            repeatY = mustNumber(
                "Please input your data col: ", dataType="int")
            for i in range(repeatY):
                tempVector = []
                for k in range(repeatX):
                    os.system('cls')
                    print("Current matrix:")
                    printMatrix(matrixPR, 2)
                    print(f"Current vector:\n{tempVector}")
```

```
                    tempInput = 0
                    tempInput = mustNumber(f"Please input the value of ({i},
{k}): ", errMsg=(
                        f"Current vector:\n{tempVector}"))
                    tempVector.append(tempInput)
                matrixPR.append(tempVector)

            for i in range(repeatX):
                os.system('cls')
                print("Completed matrix:")
                printMatrix(matrixPR, 2)
                print(f"Current output vector:\n{vectorPR}")
                tempInput = mustNumber(f"Please input the output number {i}:
", loopMsg=(
                    f"Completed Matrix:\n{matrixPR}\nCurrent output
vector:\n{vectorPR}"))
                vectorPR.append(tempInput)

            os.system('cls')
            print("Completed matrix:")
            printMatrix(matrixPR, 2)
            print(f"Current output vector:\n{vectorPR}")

        elif (isEqRegressor):
            inputX = 0
            inputY = 0
            data = []
            vectX = []
            vectY = []

            while True or (len(data) < 2):
                breaking = False

                os.system('cls')
                print("Your Data (x, y):")
                printMatrix(data, roundV=2)

                currVect = []
                inputX = mustNumber(
                    "Please input your x-data\nInput 'x' to stop (min. 2
contents).\nData: ", dataType="float", accept=['x'])
                if inputX == 'x':
                    breaking = True
                    if (len(data) >= 2):
                        break

                currVect.append(inputX)
```

```
                inputY = mustNumber(
                    "Please input your y-data\nInput 'x' to stop (min. 2
contents).\nData: ", dataType="float", accept=['x'])
                if inputY == 'x':
                    breaking = True
                    if (len(data) >= 2):
                        break

                currVect.append(inputY)

                if (not breaking):
                    data.append(currVect)
                    vectX.append(inputX)
                    vectY.append(inputY)

            power = mustNumber("Please input your power rate: ",
dataType="unsigned")

            matrix, vector = PowRegProcess(inputVectX=(vectX),
inputVectY=(vectY), power=(power+1))
            matrixPR = copy.deepcopy(matrix)
            vectorPR = copy.deepcopy(vector)
            convertToFunc = True

    if ((isLinEqSlving) or (isEqRegressor)):
        if (isLinEqSlving and isRootFinding):
            convertToFunc = True

        if (inputManual or isEqRegressor):
            checkConfig(len(vectorPR))

            guess = 0 # Fill the guess
            if (isLinEqSlving and inputManual):
                guess = []

                for i in range(len(vectorPR)):
                    os.system('cls')
                    print(f"Current guess: {vectorPR}")
                    tempInput = mustNumber(f"Please input the guess number
{i}: ", loopMsg=(
                        f"Current guess: {vectorPR}"))
                    guess.append(tempInput)

            func, gaussSdRes, gaussSdIter = GauSedProcess(matrixPR, vectorPR,
convertFunc=(convertToFunc), guess=(guess), errTol=(errTol))
        else:
            checkConfig(len(inVector))
```

```
            func, gaussSdRes, gaussSdIter = GauSedProcess(inMatrix, inVector,
convertFunc=(convertToFunc))

        fixedFunction = func
        realFunction  = func

        gaussSeidRes = copy.deepcopy(gaussSdRes)
        gaussSeidIter = gaussSdIter

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

    if (isOptimization): # Differentiate for optimization
        fixedFunction = diff(fixedFunction, x)

    if (isRootFinding):
        if (list(manualInput)[0]):
            guessOne = mustNumber("Please input 1st guess (for NR and SM): ",
dataType="float")
            guessTwo = mustNumber("Please input 2nd guess (for SM)       : ",
dataType="float")

            inputMin = mustNumber("Please input minimum restriction: ",
dataType="float")
            inputMax = mustNumber("Please input maximum restriction: ",
dataType="float")

            newtRhRes, newtRIter, secnMRes, secnMIter =
SrRootProcess(fixedFunction, intervalMin=(inputMin), intervalMax=(inputMax),
guessOne=(guessOne), guessTwo=(guessTwo), errTol=(errTol))

        else:
            newtRhRes, newtRIter, secnMRes, secnMIter =
SrRootProcess(fixedFunction)

        newtRaphRes = copy.deepcopy(newtRhRes)
        newtRaphIter = newtRIter
        secnMthdRes = copy.deepcopy(secnMRes)
        secnMthdIter = secnMIter

    # print("Fix", fixedFunction, "Real", realFunction, sep="\n")
    analysis(fixedFunction=(fixedFunction), realFunction=(realFunction),
gaussSeidIter=(gaussSeidIter), gaussSeidRes=(gaussSeidRes), newtRaphIter=(
        newtRaphIter), newtRaphRes=(newtRaphRes), secnMthdIter=(secnMthdIter),
secnMthdRes=(secnMthdRes), isRootFinding=(isRootFinding),
```

```
isLinEqSlving=(isLinEqSlving), isEqRegressor=(isEqRegressor),
manualInput=(inputManual), x_data=(matrixPR), y_data=(vectorPR),
isOptimization=(isOptimization))

# DISPLAY --- [Interface]
# 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 = [
    "Please choose your input:",
    "From Input file",
    "From manual input"
]

options_optimization = [
    "Do you wanna use optimization?",
    "Yes",
    "No"
]


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

    # Check selection validity
    if (chosen != "") and (chosen in options):
        # Pick Regress = Pick Gauss
        # if (chosen == options_type[3] and options_type[2] in options):
        #     selected.append(options_type[2])
        #     options.remove(options_type[2])

        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_optim():
    options, selected = optionDisplayer(options_optimization,
multiChoice=(False))

    decode = True if selected[0] == "Yes" else False if selected[0] == "No"
else "Error"

    # print(decode)
    return decode

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):
```

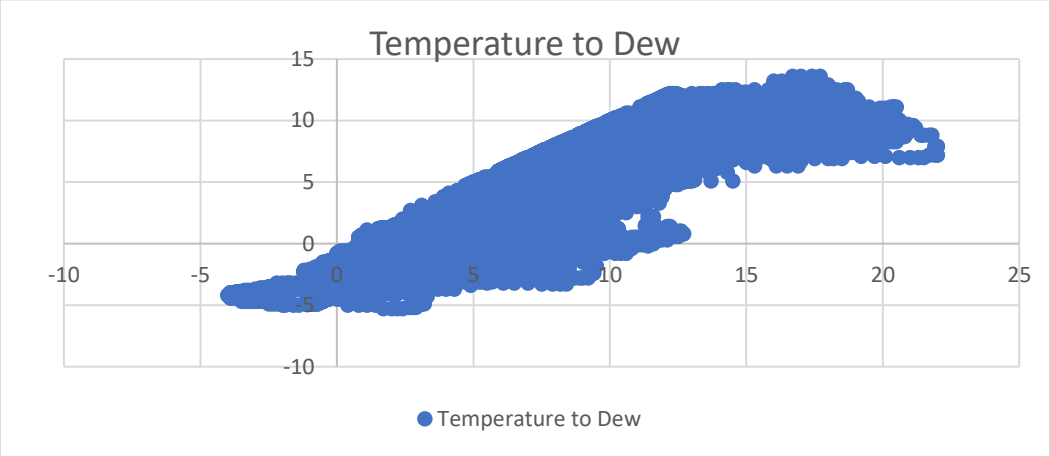
```
        print("You're choosing: Matrix --[GS]--> Function --[NR, SM]--> Root")
    elif (isRootFinding and isEqRegressor):
        print("You're choosing: (there is no supporting method for this one)")
    elif (isLinEqSlving and isEqRegressor):
        print(
            "You're choosing: Dataset --> 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()

    isOptimiziation = False
    if (isRootFinding):
        isOptimiziation = u_interface_optim()

    execute(isRootFinding=(isRootFinding), isLinEqSlving=(
        isLinEqSlving), isEqRegressor=(isEqRegressor),
manualInput={manualInput}, isOptimization=(isOptimiziation))


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))

try:
    # Data Validity
    checkValidity()

    # Execute
    u_interface()
except:
    print("\nTerminated by Error")
```

C. ANALISIS

I. Hasil Awal Simulasi

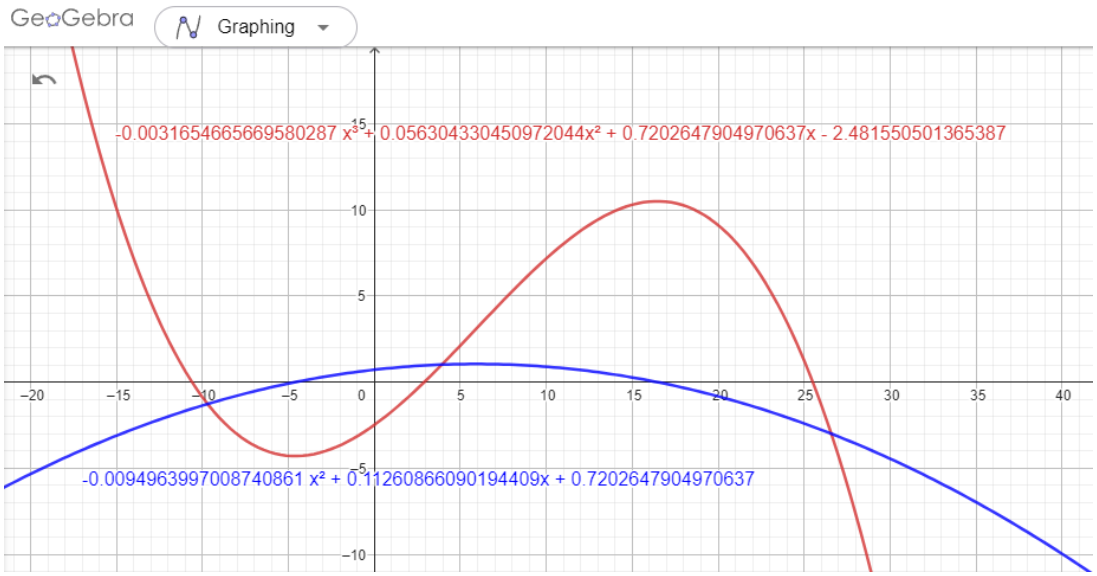
Pada folder laporan ini, terdapat suatu *file* dengan format .csv yang berisikan data set yang digunakan untuk dianalisis dalam laporan ini. Pada *file* tersebut, diambil 9000 pasangan data pertama sebagai sampel pada tahun 2017. Sisanya, pada tahun 2018 hingga 2019 diabaikan dalam laporan ini sebagai batasan percobaan. Apabila dilakukan *plotting* secara langsung, diperoleh *scatter plot* sebagai berikut.



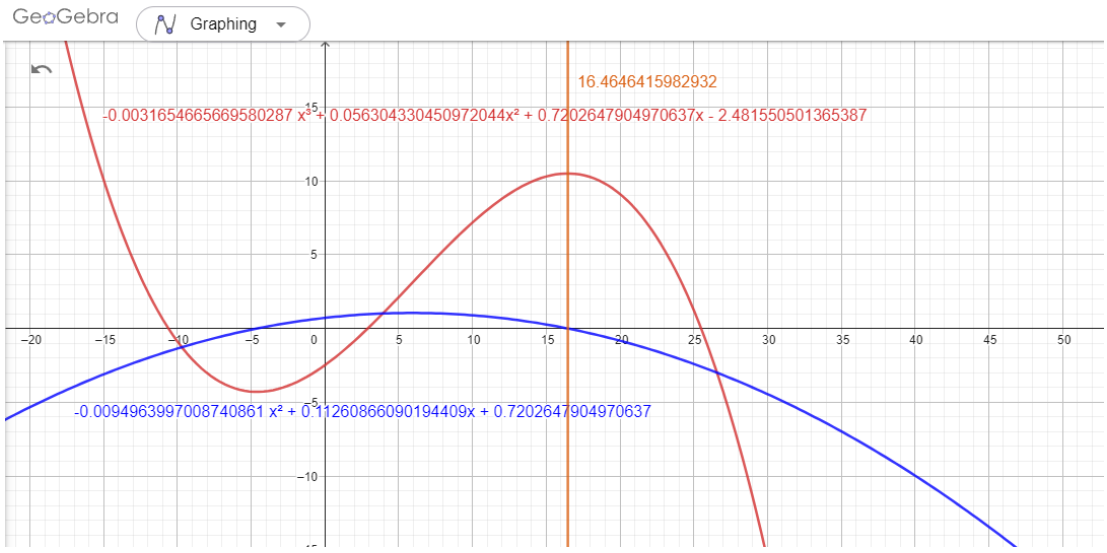
Data ini kemudian diregresi dengan polinomial berderajat tiga yang kemudian langsung dilakukan optimasi terhadapnya sebagai berikut.

```
Start to evaluate Newton Raphson
Stopped by the exact currentValue found
Newton Raphson Stopped! Last currentValue: 16.4646415982932

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: 16.4646415982932

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[-2.481550501365387, 0.7202647904970637, 0.056304330450972044, -0.0031654665669580287]
with 1000 iterations.

Function to be optimized:
-0.0031654665669580287*x**3 + 0.056304330450972044*x**2 + 0.7202647904970637*x - 2.481550501365387

Translated into Power Regressor in a function of:
-0.009496399700874086l*x**2 + 0.11260866090194409*x + 0.7202647904970637

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
16.4646415982932
with 5 iterations.
Max/min Value: 10.5121262297254
Secant Method  resulted in:
16.4646415982932
with 6 iterations.
Max/min Value: 10.5121262297254

[Evaluation completed]
```

Hasil operasi ini diperoleh dengan melakukan seleksi kepada semua fitur yang ada pada program, mengambil data dari *file* Input dan mengaktifkan fitur optimasi pada hasil akhirnya. Sesuai pada hasil di atas, dapat dilihat bahwa terdapat dua persamaan yang dihasilkan, yaitu persamaan regresi dan diferensial dari persamaan regresi yang hendak diperlakukan dengan optimasi. Kedua persamaan itu dapat divisualisasikan pada GeoGebra sebagai berikut.



Gambar ini menunjukkan bahwa persamaan dengan warna merah merupakan persamaan regresi, sementara persamaan dengan warna biru merupakan persamaan regresi yang didiferensiasi untuk menemukan titik optimum dari persamaan regresi. Dapat dilihat sesuai pada hasil program, ditemukan nilai optimum berupa nilai maksimum grafik yang dapat divisualisasikan pula dengan grafik berikut.



Jika diperhatikan lebih dalam, sebenarnya grafik tersebut juga memiliki nilai minimum pada sumbu-x (temperatur) negatif. Hal ini tidak akan dapat ditemukan oleh program mengingat program diatur sedemikian rupa untuk menolak nilai x di bawah 10. Sebagai metode pembuktian, maka dilakukan operasi program ulang tetapi dengan tebakan yang lebih rendah

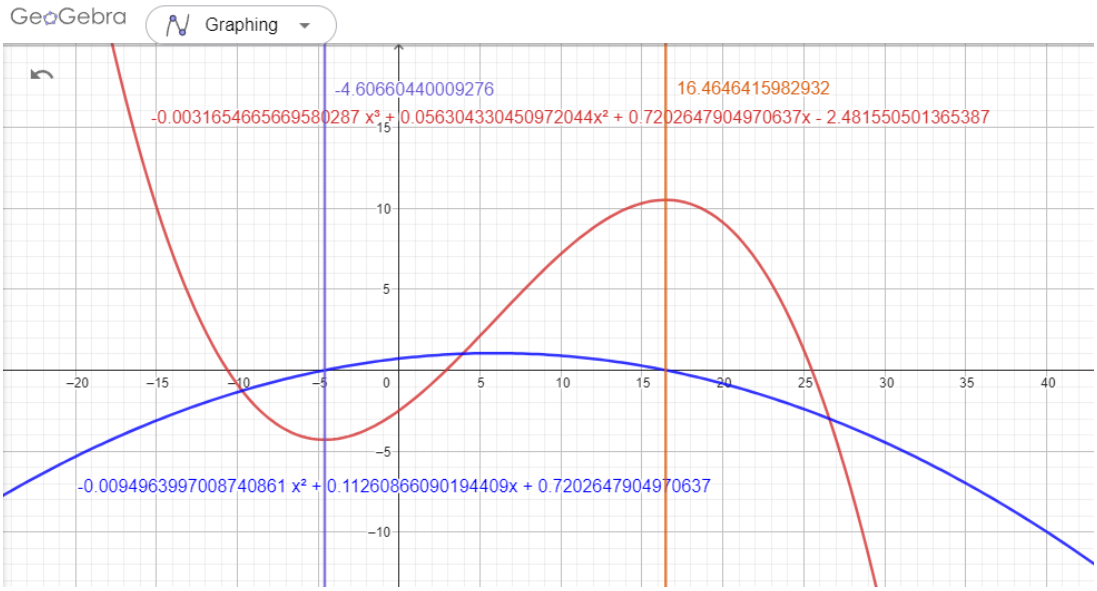
supaya dapat ditemukan akar lainnya. Adapun nilai yang lebih rendah yang dipilih dengan kriteria jarak yang dekat dengan akar kedua adalah -5 dan -6. Berikut adalah hasil dari program yang dijalankan.

```
Start to evaluate Newton Raphson
Stopped by the restricted interval exceeded

Start to evaluate Secant Method
Stopped by the restricted interval exceeded

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[-2.481550501365387, 0.7202647904970637, 0.056304330450972044, -0.0031654665669580287]
with 1000 iterations.

Function to be optimized:
-0.0031654665669580287*x**3 + 0.056304330450972044*x**2 + 0.7202647904970637*x - 2.481550501365387

Translated into Power Regressor in a function of:
-0.009496399700874086l*x**2 + 0.11260866090194409*x + 0.7202647904970637

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
10
with 1 iterations.
Max/min Value: 7.18606388174443
Secant Method  resulted in:
10
with 1 iterations.
Max/min Value: 7.18606388174443

[Evaluation completed]
```

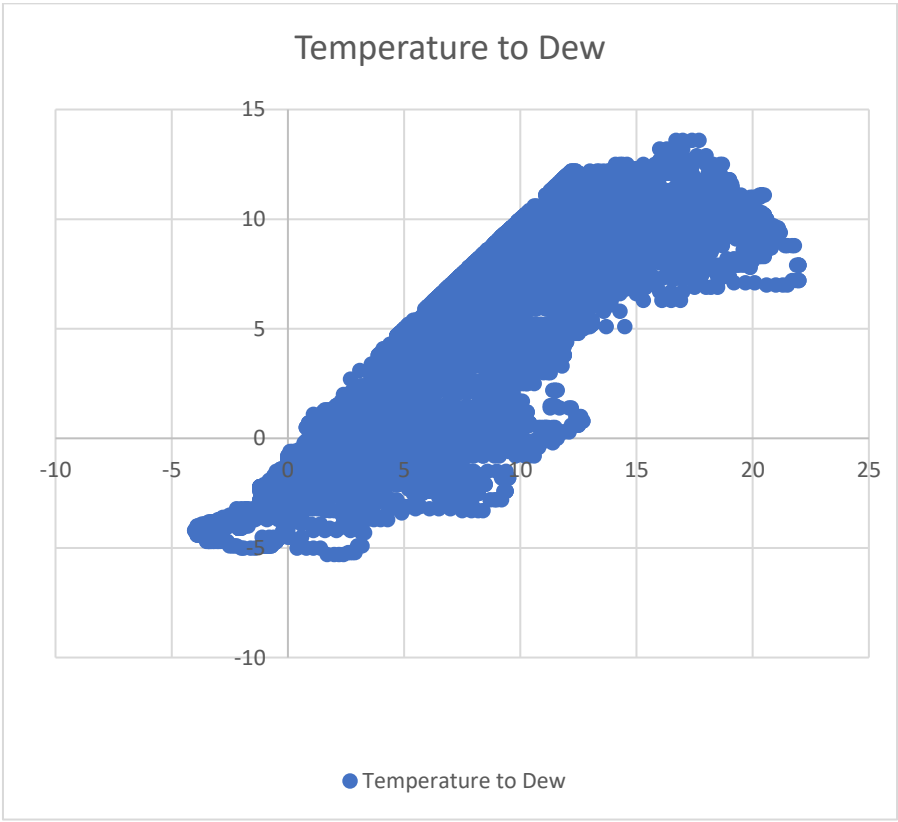
Seperti yang dapat dilihat, hasil tersebut tidak menemukan akar kedua melainkan langsung berhenti pada titik kekangan, yaitu nilai 10 Celsius dengan titik kekangan sekitar 7.18 Celcius. Jika kekangan diabaikan, maka hasil akhir program akan menjadi seperti berikut.

```
Start to evaluate Newton Raphson
Stopped by the exact currentValue found
Newton Raphson Stopped! Last currentValue: -4.60660440009276

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: -4.60660440009276

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[-2.481550501365387, 0.7202647904970637, 0.056304330450972044, -0.0031654665669580287]
with 1000 iterations.

Function to be optimized:
-0.0031654665669580287*x**3 + 0.056304330450972044*x**2 + 0.7202647904970637*x - 2.481550501365387

Translated into Power Regressor in a function of:
-0.009496399700874086l*x**2 + 0.11260866090194409*x + 0.7202647904970637

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
-4.60660440009276
with 5 iterations.
Max/min Value: -4.29525941464821
Secant Method  resulted in:
-4.60660440009276
with 6 iterations.
Max/min Value: -4.29525941464821

[Evaluation completed]
```

Maka, sesuai pada simulasi tersebut, ditemukan akar lain yang apabila divisualisasikan dengan grafik pada GeoGebra menjadi seperti berikut.



Dengan demikian, dapat didefinisikan bahwa grafik tersebut menunjukkan titik maksimum dan minimum dari perubahan titik embun yang dipengaruhi oleh tingginya temperatur udara. Sebagai pemudah pembanding, maka *scatter plot* sebelumnya juga ditampilkan kembali dalam halaman ini.



Secara visual, dapat dilihat bahwa grafik tersebut benar-benar menampilkan adanya representasi kondisi dari data yang ditampilkan pada *scatter plot* tersebut.

II. Variasi Ukuran Matriks

Sebagai pembanding, maka diambil beberapa data-data tambahan untuk melakukan analisis konsistensi pada hasil. Maka, diambil lagi 7 macam sampel data set dengan ukuran berbeda dan/atau derajat polinomial berbeda pada tahun yang sama dengan variasi kondisi seperti berikut:

1. Derajat 5 (berbeda) dengan data set 9000 pada waktu yang sama
2. Derajat 3 (sama) dengan data set 5000 data di waktu setelahnya
3. Derajat 5 (berbeda) dengan data set 5000 data di waktu setelahnya
4. Derajat 3 (sama) dengan data set tergabung 9000 + 5000
5. Derajat 5 (berbeda) dengan data set tergabung 9000 + 5000
6. Derajat 3 (sama) dengan data set terpisah sebanyak 6000
7. Derajat 5 (berbeda) dengan data set terpisah sebanyak 6000

Berikut merupakan hasil dari operasi program terhadap ketiga kondisi lain tersebut.

1. Derajat 5 (berbeda) dengan data set 9000 pada waktu yang sama

```
Start to evaluate Newton Raphson
Stopped by the tolerated round currentValue
Newton Raphson Stopped! Last currentValue: 16.5007318335391

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: 16.5007318335391

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[-2.5454709822472776, 0.7367288259124671, 0.05636244986390126, -0.002747658372284862, -7.419464186273291e-05, 2.6463730974955286e-06]
with 1000 iterations.

Function to be optimized:
2.6463730974955286e-6*x**5 - 7.419464186273291e-5*x**4 - 0.002747658372284862*x**3 + 0.05636244986390126*x**2 + 0.7367288259124671*x - 2.5454709822472776

Translated into Power Regressor in a function of:
1.3231865487477643e-5*x**4 - 0.0002967785674509316*x**3 - 0.008242975116854586*x**2 + 0.1127248997278025*x + 0.7367288259124671

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
16.5007318335391
with 4 iterations.
Max/min Value: 10.3495594068712
Secant Method  resulted in:
16.5007318335391
with 7 iterations.
Max/min Value: 10.3495594068712

[Evaluation completed]
```

2. Derajat 3 (sama) dengan data set 5000 data di waktu setelahnya

```
Start to evaluate Newton Raphson
Stopped by the exact currentValue found
Newton Raphson Stopped! Last currentValue: 21.6802904790573

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: 21.6802904790573

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[-2.027116143834799, 1.024821333039374, -0.004213580855433127, -0.0005972023556633316]
with 1000 iterations.

Function to be optimized:
-0.0005972023556633316*x**3 - 0.004213580855433127*x**2 + 1.024821333039374*x - 2.027116143834799

Translated into Power Regressor in a function of:
-0.0017916070669899995*x**2 - 0.008427161710866254*x + 1.024821333039374

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
21.6802904790573
with 6 iterations.
Max/min Value: 12.1249898299856
Secant Method  resulted in:
21.6802904790573
with 7 iterations.
Max/min Value: 12.1249898299856

[Evaluation completed]
```

3. Derajat 5 (berbeda) dengan data set 5000 data di waktu setelahnya

```
Start to evaluate Newton Raphson
Stopped by the exact currentValue found
Newton Raphson Stopped! Last currentValue: 21.4222791986270

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: 21.4222791986270

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[-3.2239349985730086, 1.181371123945621, -0.004109774040607729, -0.0008415839850140953, -1.5916695123765272e-05, 7.400321378650175e-07]
with 1000 iterations.

Function to be optimized:
7.400321378650175e-7*x**5 - 1.5916695123765272e-5*x**4 - 0.0008415839850140953*x**3 - 0.004109774040607729*x**2 + 1.181371123945621*x - 3.2239349985730086

Translated into Power Regressor in a function of:
3.700160689325087e-6*x**4 - 6.366678049506108e-5*x**3 - 0.0025247519550422866*x**2 - 0.008219548081215458*x + 1.181371123945621

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
21.4222791986270
with 6 iterations.
Max/min Value: 11.9107171823620
Secant Method  resulted in:
21.4222791986270
with 7 iterations.
Max/min Value: 11.9107171823620

[Evaluation completed]
```

4. Derajat 3 (sama) dengan data set tergabung 9000 + 5000

```
Start to evaluate Newton Raphson
Stopped by the exact currentValue found
Newton Raphson Stopped! Last currentValue: 21.3602703897247

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: 21.3602703897247

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[-2.478275510552782, 1.0442369098954205, -0.002779662396953709, -0.0006761392584320557]
with 1000 iterations.

Function to be optimized:
-0.0006761392584320557*x**3 - 0.002779662396953709*x**2 + 1.0442369098954205*x - 2.478275510552782

Translated into Power Regressor in a function of:
-0.002028417775296167*x**2 - 0.005559324793907418*x + 1.0442369098954205

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
21.3602703897247
with 6 iterations.
Max/min Value: 11.9690956653234
Secant Method  resulted in:
21.3602703897247
with 7 iterations.
Max/min Value: 11.9690956653234

[Evaluation completed]
```

5. Derajat 5 (berbeda) dengan data set tergabung 9000 + 5000

```
Start to evaluate Newton Raphson
Stopped by the tolerated round currentValue
Newton Raphson Stopped! Last currentValue: 21.1812660021810

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: 21.1812660021810

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[-2.6741327930514727, 0.9804729068488784, 0.017701272875436876, -0.0016735972517548442, -1.9472058708776263e-05, 1.2543298822465 61e-06]
with 1000 iterations.

Function to be optimized:
1.2543298822465 61e-6*x**5 - 1.9472058708776263e-5*x**4 - 0.0016735972517548442*x**3 + 0.017701272875436876*x**2 + 0.9804729068488784*x - 2.6741327930514727

Translated into Power Regressor in a function of:
6.271649411232805e-6*x**4 - 7.7888234835105052e-5*x**3 - 0.005020791755264 5326*x**2 + 0.035402545750873752*x + 0.9804729068488784

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
21.1812660021810
with 6 iterations.
Max/min Value: 11.5594719188452
Secant Method  resulted in:
21.1812660021810
with 10 iterations.
Max/min Value: 11.5594719188452

[Evaluation completed]
```

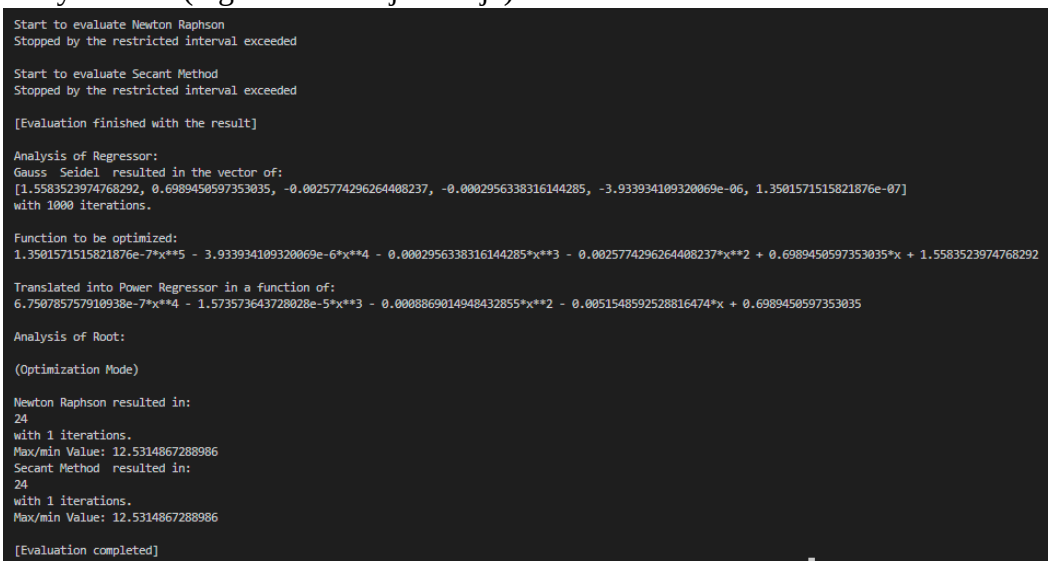
6. Derajat 3 (sama) dengan data set terpisah sebanyak 6000 [melangkahi batas]

```
Start to evaluate Newton Raphson
Stopped by the exact currentValue found
Newton Raphson Stopped! Last currentValue: 26.4605798429101

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: 26.4605798429101

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[1.8864190382441766, 0.6474428122749312, -0.002398611821155337, -0.0002478023089557088]
with 1000 iterations.

Function to be optimized:
-0.0002478023089557088*x**3 - 0.002398611821155337*x**2 + 0.6474428122749312*x + 1.8864190382441766

Translated into Power Regressor in a function of:
-0.0007434069268671264*x**2 - 0.004797223642310674*x + 0.6474428122749312

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
26.4605798429101
with 6 iterations.
Max/min Value: 12.7477546785161
Secant Method  resulted in:
26.4605798429101
with 8 iterations.
Max/min Value: 12.7477546785161

[Evaluation completed]
```

7. Derajat 5 (berbeda) dengan data set terpisah sebanyak 6000 [melangkahi batas]

```
Start to evaluate Newton Raphson
Stopped by the exact currentValue found
Newton Raphson Stopped! Last currentValue: 25.9244298922952

Start to evaluate Secant Method
Stopped by the exact value found
Secant Method Stopped!  Last Value: 25.9244298922952

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[1.5583523974768292, 0.6989450597353035, -0.0025774296264408237, -0.0002956338316144285, -3.933934109320069e-06, 1.3501571515821876e-07]
with 1000 iterations.

Function to be optimized:
1.3501571515821876e-7*x**5 - 3.933934109320069e-6*x**4 - 0.0002956338316144285*x**3 - 0.0025774296264408237*x**2 + 0.6989450597353035*x + 1.5583523974768292

Translated into Power Regressor in a function of:
6.750785757910938e-7*x**4 - 1.573573643728028e-5*x**3 - 0.0008869014948432855*x**2 - 0.005154859252881647*x + 0.6989450597353035

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
25.9244298922952
with 6 iterations.
Max/min Value: 12.5990824911281
Secant Method  resulted in:
25.9244298922952
with 7 iterations.
Max/min Value: 12.5990824911281

[Evaluation completed]
```

8. Ekstra, *output* sesungguhnya pada batas kekangan maksimum untuk data set terpisah sebanyak 6000 (digunakan derajat 5 saja).



```
Start to evaluate Newton Raphson
Stopped by the restricted interval exceeded

Start to evaluate Secant Method
Stopped by the restricted interval exceeded

[Evaluation finished with the result]

Analysis of Regressor:
Gauss  Seidel  resulted in the vector of:
[1.5583523974768292, 0.6989450597353035, -0.0025774296264408237, -0.0002956338316144285, -3.933934109320069e-06, 1.3501571515821876e-07]
with 1000 iterations.

Function to be optimized:
1.3501571515821876e-7*x**5 - 3.933934109320069e-6*x**4 - 0.0002956338316144285*x**3 - 0.0025774296264408237*x**2 + 0.6989450597353035*x + 1.5583523974768292

Translated into Power Regressor in a function of:
6.750785757910938e-7*x**4 - 1.573573643728028e-5*x**3 - 0.0008869014948432855*x**2 - 0.005154859252881647*x + 0.6989450597353035

Analysis of Root:

(Optimization Mode)

Newton Raphson resulted in:
24
with 1 iterations.
Max/min Value: 12.5314867288986
Secant Method  resulted in:
24
with 1 iterations.
Max/min Value: 12.5314867288986

[Evaluation completed]
```

III. Analisis Hasil

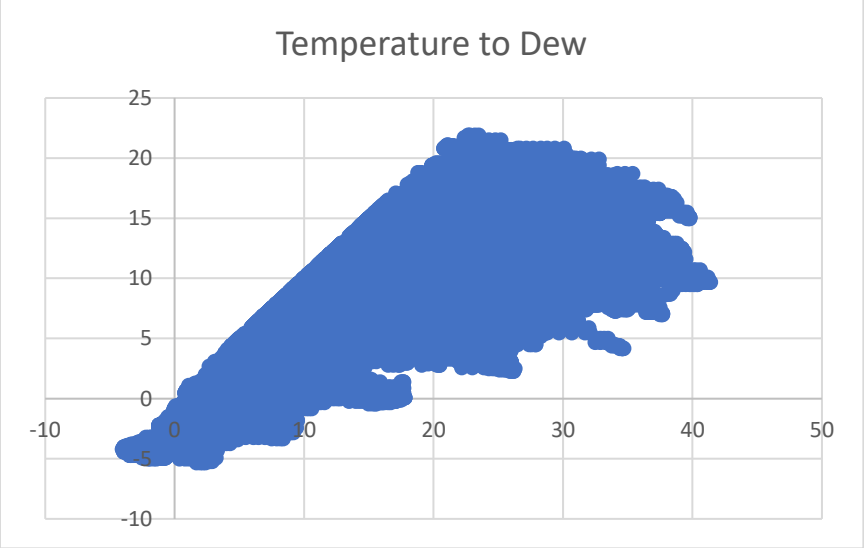
Berdasarkan apa yang telah diperoleh, dapat dikompilasi hasil-hasil operasi program tersebut sebagai berikut.

No	Derajat	Data	Temperatur Udara pada Titik Embun Maksimum (℃)	Titik Embun Maksimum (℃)
1	3	9000	16.4646415982932	10.5121262297254
2	5	9000	16.5007318335391	10.3495594068712
3	3	5000	21.6802904790573	12.1249898299856
4	5	5000	21.4222791986270	11.9107171823620
5	3	14000	21.3602703897247	11.9690956653234
6	5	14000	21.1812660021810	11.5594719188452
7	3	6000	26.4605798429101	12.7477546785161
8	5	6000	25.9244298922952	12.5990824911281
9	5	6000	24.0000000000000	12.5314867288906

Sesuai pada tabel di atas, dapat dilihat bahwa titik embun maksimum berada berkisar 10.3℃ hingga 12.6℃ dengan penyebab kenaikan titik embun hingga maksimum adalah temperatur udara yang berkisar 16.4℃ hingga 24℃. Tetapi, terdapat beberapa bentuk tidak konsisten dalam hasil ini yang dapat dilihat pada tingginya deviasi pada temperatur udara pada titik embun maksimum. Meski demikian, deviasi titik embun maksimum justru tidaklah tinggi. Berikut merupakan nilaian deviasi yang dimaksud. (Catatan: data pada nomor ke-7 dan ke-8 tidak digunakan karena melebihi kekangan)

Parameter	**Temperatur Udara** (℃)	**Temperatur Titik Embun** (℃)
Mean ($\bar{x}$)	20.37278279	11.56534957
Deviasi (s)	2.824383319	0.828288154

Pada tabel sebelumnya, dapat dilihat bahwa temperatur udara yang meskipun cukup tinggi deviasi sampelnya (hingga 2.82℃), deviasi sampel dari temperatur titik embun tergolong relatif kecil (hanya 0.83℃ saja). Berdasarkan hal tersebut, dapat dikatakan bahwa sesungguhnya temperatur maksimum dapat berupa berapa saja di antara interval 16.4℃ hingga 24℃ yang pada akhirnya tetap akan menghasilkan titik embun maksimum di sekitar 11.56℃. Adapun berikut ini merupakan grafik total keseluruhan dari 35.040 data yang diperoleh pada 2017 untuk kota İzmir, Turki sebagai perbandingan terhadap hasil percobaan.



Grafik tersebut memang menunjukkan bahwa titik maksimum dari titik embun terletak pada nilai sekitar 23℃ dengan temperatur sekitar 24℃ apabila dilakukan tebakan nilai. Namun, perlu dicatat pula bahwa nilai tersebut melebihi kekangan yang telah ditentukan sebelumnya. Selain itu, data yang diolah sebelumnya diambil secara langsung dari nilai-nilai awal data set yang diperoleh (bulan-bulan pertama pengambilan cuplikan data). Hal ini berarti bisa saja terdapat faktor cuaca pada musim yang berbeda pada bulan-bulan berikutnya yang mempengaruhi perubahan temperatur rata-rata tertentu pada waktu lainnya yang tidak diolah dalam laporan ini.

D. KESIMPULAN

Berdasarkan percobaan yang dilakukan pada laporan ini, terdapat kesimpulan yang dapat digolongkan menjadi dua, yaitu kesimpulan matematis dan kesimpulan inferensial. Berikut merupakan kesimpulan matematis dari percobaan.

1. Perubahan derajat dari derajat 3 ke derajat 5 dalam kasus ini tidak menghasilkan perubahan hasil regresi polinomial yang signifikan. Hal ini dapat dilihat pada data yang diperoleh di mana masing-masing bentuk derajat dalam variasi data yang sama hanya memiliki perbedaan hasil maksimal 0.5℃.
2. Perbedaan hasil temperatur udara pada titik embun maksimum pada masing-masing pasangan data yang berbeda tidak terlalu berdampak dengan deviasi temperatur titik embun dengan nilai deviasi serendah sekitar 0.83℃ saja.
3. Senada dengan kesimpulan ke-2, maka meskipun simpangan temperatur udara tergolong cukup besar, hal ini tidak mempengaruhi besarnya temperatur titik embun pada daerah tersebut.
4. Dengan perbandingan yang dilakukan terhadap *scatter plot* data asli, ditemukan bahwa hasil simulasi (20.36℃, 11.57℃) belum dapat merepresentasikan kondisi satu tahun hubungan antara temperatur udara dengan titik embun.

Selain itu, sesuai dengan kesimpulan-kesimpulan matematis tersebut, dapat diambil kesimpulan inferensial seperti berikut ini.

1. Pendekatan regresi polinomial terhadap data menunjukkan bahwa derajat persamaan tidaklah diprioritaskan. Hal ini berarti model matematika yang digunakan untuk merepresentasikan hubungan antara temperatur udara terhadap titik embun dapat dinyatakan dalam hubungan kuadrat-parabola tertutup sederhana saja (semakin tinggi temperatur, semakin tinggi titik embun hingga titik tertentu hingga akhirnya menurun kembali).
2. Titik embun maksimum tidak secara kaku diakibatkan oleh beberapa titik temperatur saja. Pada data yang diperoleh, 16.46℃ - 24.00℃ merupakan interval yang menghasilkan titik maksimum yang tak jauh berbeda. Hal ini menyatakan bahwa interval temperatur tersebut adalah temperatur yang dapat mengakibatkan titik embun tertinggi pada daerah tersebut.
3. Perlu dilakukan simulasi yang lebih masif untuk mampu mendefinisikan pendekatan terhadap data asli yang lebih baik. Hal ini berarti kekangan harus diperlonggar dan data yang dioperasikan harus ditingkatkan jumlahnya untuk menghasilkan titik optimum yang lebih akurat.