2025 IMEKO TC-6 International Conference on
Metrology and Digital Transformation - **M4DConf 2025**
Benevento, Italy, September 3-5, 2025

# Thermometry Machine Learning Model for Digitized Metrological Calibration of Platinum Resistance Thermometer

Oqab. N. Alotaibi[1], Rakan O. Alnefaie[1], Arwa K. Alrushud[1], Fahad A. AlMuhlaki[1], Rayan A. AlYousefi [1], Saad A. Bin Qoud[1], I. AlFaleh[1], N. Qahtani[1], A. El-Matarawey[2]

1. NMCC-SASO- Saudi Arabia
2. National Institute of Standards (NIS-Egypt)

*Abstract* – **Temperature measurements rely on various types of thermometers, including but not limited to Platinum Resistance Thermometers (PRTs), thermocouples, and radiation thermometers. Among these, resistance thermometers are considered highly reliable for sensitive temperature measurements. To ensure the accuracy and precision of measurement results, it is essential to consider factors that affect either the temperature value itself (after conversion from ohms) or the uncertainty estimation when using resistance thermometers.**

**One critical factor is the interpolation error that arises when converting resistance values to temperature using the ITS-90 equations. Discrepancies in the temperature values obtained through these methods can impact measurement reliability. Therefore, this study aims to develop a robust Python-based algorithm for calibrating PRTs with minimal errors, thereby reducing the impact on measurement uncertainty.**

**The study will provide an open-source, step-by-step algorithm as part of the global digital transformation trend. This algorithm will serve as a valuable resource for researchers and practitioners seeking to enhance the reliability and accuracy of temperature measurements.**

**Keyword -** Temperature, Resistance, control, Python, fixed point, ITS-90.

## I. INTRODUCTION

The National Metrology and Calibration Center (NMCC) at the Saudi Standards, Metrology & Quality Organization (SASO) includes the Thermometry Laboratory, a prominent facility dedicated to the enhancement, advancement, and preservation of national standards in Thermometry under the International Temperature Scale of 1990 (ITS-90) [1]. Additionally, the Thermometry Laboratory provides testing and calibration services to a wide range of industries, including medical, industrial agriculture, pharmaceuticals, and scientific sectors. This underscores the vital role that National Metrology Institutes (NMIs) play in maintaining quality infrastructure. According to the Bureau International des Poids et Mesures (BIPM), "Metrology" refers to the science of measurement. It is essential that measurements of physical properties, such as temperature, are conducted using reliable stabilization systems. Researchers, scientists, equipment developers, and manufacturers have a keen interest in measuring various mediums. In the field of metrology at National Metrology Institutes, the primary focus is on achieving accuracy and precision within a few millikelvins (mK). Industrial Platinum Resistance Thermometers (IPRTs), also known as Pt-100s, utilize platinum as the sensing resistor due to its high sensitivity to temperature, favorable temperature coefficient, ability to withstand high temperatures (up to 962 °C in certain cases), and resistance to chemical corrosion, particularly oxidation. Platinum resistance thermometers are calibrated at specific fixed temperature points within their range, as per ITS-90. To simplify the process for users, calibration outcomes are converted from resistance to temperature. The ITS-90 calculation can be used for unit conversion when calibration is performed at fixed points [2, 3]. In recent years, digital technologies have seen significant advancements, including concepts like big data, cloud computing, artificial intelligence (AI), and machine learning (ML). ML, deep learning, and neural networks have been effectively integrated into the field of metrology [4, 5, 6, 7, 8, 9, 10]. The exponential growth in computing and storage capacities, coupled with faster data exchange speeds and the affordable availability of versatile sensors, has created new opportunities for improving metrology tools, such as Python programming language. Python is one of the most widely used programming languages, serving various purposes, including machine learning, web development, desktop applications, and providing a robust environment for parallel mathematics. Its syntax is straightforward and user-friendly, making it an excellent choice for many applications. Python can be easily adapted for machine learning (ML), artificial intelligence (AI), descriptive and inferential statistics, and advanced mathematics, delivering great results and accuracy even

with limited resources and time. Python offers a wide range of libraries tailored to specific tasks based on the application area. In our case (Advanced Mathematics), we will use libraries that focus on algebra, matrices, mathematics, visualizations, exploratory data analysis (EDA), statistics, data pipelines, and data containers.

## II. ITS-90 TEMPERATURE SUB-RANGES

In this study we selected a wide subrange for investigations form -40 ºC up to 660 ºC that split into two main subranges. The resistance ratio and general deviation function is given by equation 1 and 2 respectively [1, 2, 3, 11, 12, 22].

$$W(T_{90}) = R(T_{90})/R(273.16\,K) \quad (1)$$

$$W(T_{90}) - W_r(T_{90}) = a[W(T_{90}) - 1] + b[W(T_{90}) - 1]^2 + c[W(T_{90}) - 1]^3 + d[W(T_{90}) - W_{Al}]^2 \quad (2)$$

The Mercury Triple Point (-38.8344 °C) to the Gallium Melting Point (29.7646 °C) with c, d = 0

$$W(T_{90}) - W_r(T_{90}) = a[W(T_{90}) - 1] + b[W(T_{90}) - 1]^2 \quad (3)$$

The constants a and b are determined from the calibration of thermometer at mercury triple point and gallium melting point. The Water Triple Point 0 °C to the Aluminum Freezing Point (660.323 °C) with d = 0.

$$W(T_{90}) - W_r(T_{90}) = a[W(T_{90}) - 1] + b[W(T_{90}) - 1]^2 + c[W(T_{90}) - 1]^3 \quad (4)$$

The constants a, b and c are determined from the calibration of thermometer at tin freezing point, zinc freezing point and aluminum freezing point.

## III. MACHINE LEARNING MODEL

### III. A. Python environment

The Python ecosystem provides a comprehensive suite of libraries for scientific computing, data analysis, and machine learning, often utilized within the interactive environment of Jupyter Lab [13]. At its core, NumPy [14] serves as a fundamental library for numerical computations, offering support for multi-dimensional arrays, matrices, and a wide range of mathematical operations. Built on top of NumPy, Pandas [15] provides high-level data manipulation tools, including Series and DataFrame structures, enabling efficient management and analysis of tabular data. For visualization, Matplotlib [16] allows the creation of complex static plots, while Plotly [17] offers interactive and dynamic visualizations. SciPy [18] extends NumPy's functionality by providing advanced tools for scientific computing, such as optimization, integration, and linear algebra. Seaborn [19], based on Matplotlib, simplifies the creation of visually appealing statistical graphics through a high-level interface. Statsmodels [20] supports statistical modeling, hypothesis testing, and data exploration with extensive result statistics for various estimators. Finally, Scikit-learn [21] stands out as a powerful library for machine learning, offering efficient tools for tasks like classification, regression, clustering, and dimensionality reduction through a consistent API. Together, these libraries form an indispensable toolkit for algorithm development, data analysis, and visualization in Python.

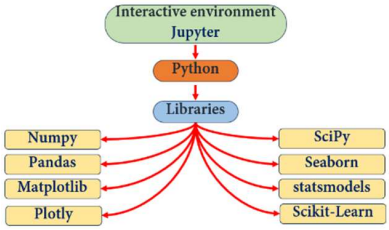


*Fig. 1. Interactive environment of Python with an integrated library*

### III. B. MODEL ARCHITECTURAL.

The workflow for data acquisition, processing, and machine learning model development can be described in a structured manner as shown in figure 2.

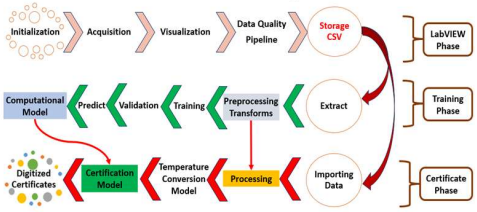


*Fig. 2. Model architectural and data pipeline*

- *Initialization*: The system initializes with parameters P and configurations C: System← {P, C}, the parameters here are kind of equipment such as ASL thermometry bridge, standard resistor bath with specific configuration as low gain and 1 mA current etc.

- *Acquisition*: Data Frame DF is acquired from SPRT as a time series versus resistance ratio, live standard resistor, oil bath temperature and macroscopic-microscopic feature engineering.

$$DF = \{x_1, x_2, \dots, x_n\}, x_i \in \mathbb{R}^m \quad 5$$

where n is the number of samples, and m is the dimensionality of each sample.

- *Visualization:* Raw data DF["Ratio"] is visualized using functions $V(DF["Ratio"]): \mathbb{R}^{n\times m}$ For example, scatter plots or histograms are generated to explore distribution.

- *Data Quality Pipeline:* Data quality metrics Q(DF) are computed to ensure integrity

$$Q(D) = \{\mu, \sigma, missing\_values\} \quad 6$$

Where $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$ is mean and standard deviation of the trained data respectively. Missing values are flagged and handled for example dropped NA, fill forward or backward using live synchronization.

- *Storage (CSV):* Validated data DF is stored in CSV format, DF→CSV(DF).

- *Extract:* Data is extracted from CSV files for further processing as $DF' = Extract(CSV(DF))$

- *Preprocessing Transforms:* Preprocessing involves transformations $T(DF') = \{Z, E, S\}$ where Z is a zero-mean normalization $z_i = (\frac{X_i - \mu}{\sigma})$, E is Encoding categorical variables and S is scaling features to a range [0,1]

- *Processing:* Feature extraction F(DF′) transforms data into a feature matrix $X = F(D'), X \in \mathbb{R}^{n \times k}$ where k is the number of engineered features.

- *Training:* A machine learning model M is trained on features matrix X and target vector Resistance(R), as $M = Train(X, R; \theta)$, where θ is the model parameters optimized via a loss function L

$$\theta^* = arg\ min_\theta L(M(X;\theta), R) \quad 7$$

- *Validation:* The trained model M is validated using a separate validation set $X_{val}$ and $R_{val}$, Common metrics include accuracy, mean squared error (MSE), etc.

## IV. CASE STUDY

As mentioned before importing the pythonic libraries into the environment (figure 3). for simplicity the libraries were aliased be short name for example python know NumPy by its new name np. The lines of codes were introduced in as a screenshot from the program in order to visualize the different in colors for each piece of codes.

```
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sn
from matplotlib import pyplot as plt
import scipy as sp
import statsmodels as sm
from sklearn.linear_model import LinearRegression
```

*Fig 3. Loading the required libraries into the python environment*

Reading step used Pandas library (pd) that read the csv (comma-separated-values) file, put the data into a container called data frame (df) and display the first 5 rows of the data, Figure 4. The reading step is iterable for all calibration files for other fixed points, water triple point file will be introduced as an example and other files will be the same.

```
# Import the triple point of water data from CSV file
df = pd.read_csv('water_triple_point.csv')
df.head(5)
```

*Fig 4. Loading the required libraries into the python environment*

The output will be in the form of table called df that includes data about the time of triggering the measurement, index of data value that start in python with zero and finally the value itself. for simplicity in our case, we interesting only in two columns the value and its index number.

*Table 1. Reading data from water triple point file*

| Index | Value/Ω |
|---|---|
| 0 | 99.392992 |
| 1 | 99.393673 |
| 2 | 99.392815 |
| 3 | 99.393372 |
| 4 | 99.393572 |

Extracting step are using matplotlib, SciPy and statsmodel libraries for manipulating, visualization, performing normal distribution fitting and finally recalling the median with the required confidence interval after doing the required correction due to the self-heating.

```
#----------Normal Distribution plot----------
plt.figure(figsize=(10,8))
count, bins, ignored = plt.hist(df, 90, density=True,
                                edgecolor="black")
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
               np.exp( - (bins - mu)**2 / (2 * sigma**2)),
         linewidth=2, color='r')
plt.axvline(mu, color = 'red', linestyle = '--',linewidth=2)
plt.axvline(np.quantile(df,0.05), color = 'red',
            linestyle = '--',linewidth=2)
plt.axvline(np.quantile(df,0.95), color = 'red',
            linestyle = '--',linewidth=2)
plt.xlabel('Resistance / Ohm')
plt.ylabel('Density')
plt.rcParams.update({'font.size': 25})
plt.show()
#----------------Median----------------
Rw= np.median(df)
```

*Fig 5. Statistical treatment of the df with normal distribution fitting and obtaining the median value.*

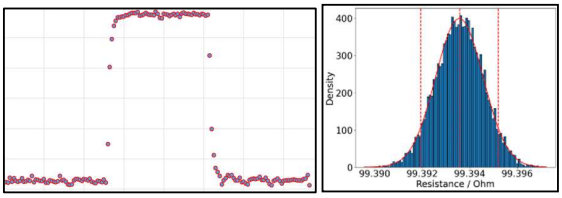


*Figure 6. Normal distribution fitting of the data.*

The best estimation of the plateau point will be stored in a new variable Rw = 99.39358 Ω.

Starting from instantiating the variables, the value for eac h file will be stored in new name that is represent on its so urce. $R_w, R_{Hg}, R_{Ga}, R_{Sn}, R_{Zn}, R_{Al}$ variables are representin g for the abs-olute resistance of the measurements at fixed points after multiplied by the value of standard resistor Te nsily $R_s = 100.00077$. Now it is the time of resistance ra tio W($T_{90}$), by apply equation 1 and stored the variables in a new names $W_{Hg}, W_{Ga}, W_{Sn}, W_{Zn}, W_{Al}$ (Figure 7).

```
# The value of triple point of water
Rw=99.39358;

# The value of triple point of mercury
Rhg=83.90765;

# The value of melting point of gallium
Rga=111.13158;

# The value of freezing point of tin
Rsn=188.10386;

# The value of freezing point of Zinc
Rzn=255.281564;

# The value of freezing point of Alluminium
Ral=335.46533;

# W(T90)
Wsn=Rsn/Rw;
Wzn=Rzn/Rw;
Wal=Ral/Rw;
Whg=Rhg/Rw;
Wga=Rga/Rw;
```

*Fig 7. Instantiating the resistance variable at each fixed point.*

```
# Chicking the criteria of ITS-90
Wga>=1.11807

True

# Chicking the criteria of ITS-90
Whg>=0.844235

True
```

*Figure 8. Checking the criteria of ITS-90*

As described in the text of ITS-90, to be considered a suitable platinum resistance thermometer, it is necessary for the thermometer to be constructed using pure and strain-free platinum, and it must meet one of the two conditions outlined as W (29.7646 °C) ≥ 1.118 07 or W (-38.8344 °C) ≥ 0.844 235.

In this case, the thermometer satisfies one condition of the criterion and the algorithm will continue for next step. If the thermometer fails to satisfy both criterions, the thermometer will be introduced for another algorithm based on Callendar Van Dusen equation which will be introduced in another article.

This is the longest part of algorithm and the most computational resources consumable. The advantage of python is its limited consuming operations, python is still work efficiently, fast even with low computers. the temperature step is consisting of several sub-steps, it begins with calculating the conversion constants, evaluating Wr, calculating the temperature, looping for the range of Wr against temperature, calculate the increments in temperature against 0.01 change in the value Wr and finally building table of the specific range. Return the coefficients of the system of equations based on equation (3). Solve equations in two variables a, b as shown in code block figure 9.

```
# Range from HgTP up to GaMP--------------------------------
#(Whg-0.84414211==(a4*(Whg-1))-(b4*(Whg-1)**2)
#(Wga-1.11813889==(a4*(Wga-1))-(b4*(Wga-1)**2)
A = np.array(
    [
        [(Whg-1),(Whg-1)**2],
        [(Wga-1), (Wga-1)**2]
    ],dtype=np.float64
)
b = np.array([(Whg-0.84414211), (Wga-1.11813889)], dtype=np.float64)
y = np.linalg.inv(A).dot(b)
y
```

*Figure 9. Input system of equations based on NumPy solve linear algebra function (2 X 2 Equations).*

The return constants will be aliased as $a_l$, $b_l$ for low temperature coefficients, there values are ($a_l$=-3.54604293e-04, $b_l$=6.12237423e-05)

Building table of temperature based on the following equation as shown in figure 10 and table 2.

$$\frac{T_{90}}{273.16} = B_o + \sum_{i=1}^{15} B_i \left[\frac{W_r(T_{90})^{\frac{1}{6}} - 0.65}{0.35}\right]^i \qquad 8$$

```
tl=[]
for Wt in [Y / 100 for Y in range(83, 101, 1)]:
    Wr=Wt-(al*(Wt-1))-((bl*(Wt-1)));
    W=((Wr**(1/6))-0.65)/0.35
    t=(((Bo)+(B1*W)+(B2*(W)**2)+(B3*(W)**3)+(B4*(W)**4)
        +(B5*(W)**5)+(B6*(W)**6)+(B7*(W)**7)+(B8*(W)**8)
        +(B9*(W)**9)+(B10*(W)**10)+(B11*(W)**11)+(B12*(W)**12)
        +(B13*(W)**13)+(B14*(W)**14)+(B15*(W)**15))*(273.16))-273.15;
    tl.append(t)
df_TL= pd.DataFrame(tl, columns=['Temperature'])
Y=[Y / 100 for Y in range(83, 101, 1)]
df_WtL=pd.DataFrame(Y, columns=['W'])
```

```
nL=len(df_TL['Temperature'])
incL=[]
for i in range (0, nL-1, 1):
    incr=df_TL['Temperature'][i+1]-df_TL['Temperature'][i]
    incL.append(incr)
df_incL=pd.DataFrame(incL, columns=['Increament'])
```

```
df_L=pd.concat([df_WtL, df_TL, df_incL], axis=1)
df_L.head()
```

*Figure 10. Input codes for generating table of low temperature range against resistance ratio and increment.*

*Table 2. First five rows of the generated data for low range.*

| Index | W | Temperature | Increament |
|---|---|---|---|
| 0 | 0.83 | -42.353109 | 2.476393 |
| 1 | 0.84 | -39.876716 | 2.478385 |
| 2 | 0.85 | -37.398332 | 2.480364 |
| 3 | 0.86 | -34.917968 | 2.482331 |
| 4 | 0.87 | -32.435637 | 2.484287 |

Return the coefficients of the system of equations based on equation (4). Solve equations in three variables a, b and c as shown in code block figure 11.

```
# Range from WTP up to ALFP---------------------------------
    # Wsn-1.89279768==(ah*(Wsn-1))-(bh*(Wsn-1)**2)-(ch*(Wsn-1)**3)
    # Wzn-2.56891730==(ah*(Wzn-1))-(bh*(Wzn-1)**2)-(ch*(Wzn-1)**3)
    # Wal-3.37600860==(ah*(Wal-1))-(bh*(Wal-1)**2)-(ch*(Wal-1)**3)
A = np.array(
    [
        [(Wsn-1),(Wsn-1)**2, (Wsn-1)**3],
        [(Wzn-1), (Wzn-1)**2, (Wzn-1)**3],
        [(Wal-1), (Wal-1)**2, (Wal-1)**3]
    ],dtype=np.float64
)
b = np.array([(Wsn-1.89279768), (Wzn-2.56891730),(Wal-3.37600860)],
             dtype=np.float64)
x = np.linalg.inv(A).dot(b)
x
```

*Figure 11. Input system of equations based on NumPy solve linear algebra function (3 X 3 Equations).*

The return constants will be aliased as $a_h$, $b_h$ and $c_h$ for low temperature coefficients, there values are ($a_l$= -3.09195651e-04, $b_l$= -1.28280350e-05, $c_h$ =3.25476196e-06)

Building table of temperature based on the following equation as shown in figure 12 and table 3.

$$T_{90} - 273.15 = D_o + \sum_{i=1}^{9} D_i \left[\frac{W_r(T_{90}) - 2.64}{1.64}\right]^i \qquad 9$$

```
th=[]
for Wt in [X / 100 for X in range(100, 339, 1)]:
    Wr=Wt-(ah*(Wt-1))-(bh*(Wt-1)**2)-(ch*(Wt-1)**3)
    W=(Wr-2.64)/1.64
    t=(Do+(D1*W)+(D2*(W)**2)+(D3*(W)**3)+(D4*(W)**4)+
        (D5*(W)**5)+(D6*(W)**6)+(D7*(W)**7)+(D8*(W)**8)+
        (D9*(W)**9))
    th.append(t)
df_T= pd.DataFrame(th, columns=['Temperature'])
X=[X / 100 for X in range(100, 339, 1)]
df_Wt=pd.DataFrame(X, columns=['W'])

n=len(df_T['Temperature'])
inc=[]
for i in range (0, n-1, 1):
    incr=df_T['Temperature'][i+1]-df_T['Temperature'][i]
    inc.append(incr)
df_inc=pd.DataFrame(inc, columns=['Increament'])

df_H=pd.concat([df_Wt, df_T, df_inc], axis=1)
df_H.head()
```

*Figure 12. Input codes for generating table of high* temperature range against resistance ratio and increment.

*Table 3. First five rows of the generated data for high range.*

| Index | W | Temperature | Increament |
|---|---|---|---|
| 0 | 1.00 | 0.010000 | 2.508940 |
| 1 | 1.01 | 2.518940 | 2.510865 |
| 2 | 1.02 | 5.029805 | 2.512793 |
| 3 | 1.03 | 7.542598 | 2.514723 |
| 4 | 1.04 | 10.057321 | 2.516656 |

Collect step based on Pandas, seaborn, scikit-learn and scipy, the algorithm will merge the two generated tables in one csv file, collect the conversion constants in one csv file. Plot step based on matplotlib and seaborn for good visualizations. The algorithm is plotting the linearity curve, thermometer has a response that is close to good linearity. A common approximation is to draw a "straight line curve," starting from the triple point of mercury up to freezing point of Aluminum. The Y-axis represents to the Temperature / ºC and X-axis represents to the resistance ration Rt/Rs, figure 13.

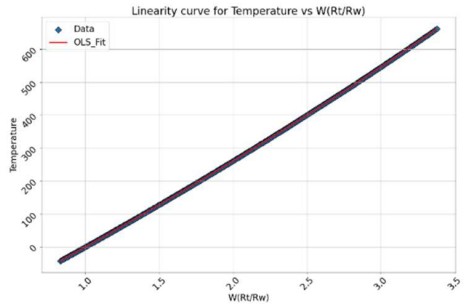


*Figure 13. Linearity curve for temperature against W.*

In validation step, bulk of codes was created in order to build a function called ***convert_R.*** The function task is to gathering the resistance value in ohm, locate its position in high or low ranges and convert it to temperature / ºC, figure 14. **If** the resistance value less than Rw, the algorithm is going to apply equation 5. **Else**, the the algorithm apply equation 6. The code is called *RPC* for Resistance - Positioning - Conversion code.

The validation mechanism starts with input the actual measured resistance and subtracting from the actual defining temperature in the text of ITS-90, figure 15. the code is called *ARTC* for Actual-Resistance-Temperature-Conversion code. The second step of validation mechanism is building table of discrepancies between $T_{90}$ and $T_{Model}$ as shown in table 4.

```
# Validation of Model
def convert_R(resistance):
    if resistance < Rw:
        Wt=resistance / Rw
        Wr=Wt-(al*(Wt-1))-((bl*(Wt-1))*(np.log(Wt)))
        W=((Wr**(1/6))-0.65)/0.35
        t=(((Bo)+(B1*W)+(B2*(W)**2)+(B3*(W)**3)+(B4*(W)**4)
        +(B5*(W)**5)+(B6*(W)**6)+(B7*(W)**7)+(B8*(W)**8)
        +(B9*(W)**9)+(B10*(W)**10)+(B11*(W)**11)+(B12*(W)**12)
        +(B13*(W)**13)+(B14*(W)**14)+(B15*(W)**15))*(273.16))
        -273.15
    else:
        Wt=resistance / Rw
        Wr=Wt-(ah*(Wt-1))-(bh*(Wt-1)**2)-(ch*(Wt-1)**3)
        W=(Wr-2.64)/1.64
        t=(Do+(D1*W)+(D2*(W)**2)+(D3*(W)**3)+(D4*(W)**4)+
        (D5*(W)**5)+(D6*(W)**6)+(D7*(W)**7)+(D8*(W)**8)+
        (D9*(W)**9))
    return t
```

*Figure 14. Algorithm Resistance-Positioning-Conversion code*

```
t_w=convert_R(Rw)
print(
    'The corresponding temperature is {} oC'
    .format(t_w)
)
```

```
The corresponding temperature is 0.009999999999990539 oC
```

```
t_w-0.01
```

```
-9.461008365629908e-14
```

*Figure 15. Actual-Resistance-Temperature-Conversion code*

*Table 4. Summary of discrepancies between $T_{90}$ and* $T_{Model}$

| $T_{90}$ ºC | $T_{Model}$ ºC | Error ($T_{Model}$ - $T_{90}$) ºC |
|---|---|---|
| -38.8344 | -38.83442960353265 | 2.96035326 x $10^{-05}$ |
| 0.0100 | 0.0099999999999053 | -9.4610083 x $10^{-14}$ |
| 29.7646 | 29.764598216023084 | 1.78397691 x $10^{-06}$ |
| 231.9280 | 231.9280732051055 | 7.32051055 x $10^{-05}$ |
| 419.5270 | 419.5269848357539 | -1.5164246 x $10^{-05}$ |
| 660.3230 | 660.3230567011886 | 5.67011886 x $10^{-05}$ |

The propagation of the error was estimated according to fitting the $T_{90}$ against the $T_{Model}$ - $T_{90}$ as shown in figure 16. The interpolating-Extrapolating errors [22] can be estimated from the following equation. $Error\ (predicted) = (3.985e-12)\ T_{90}^3 - (3.913e-09)\ T_{90}^2 + (9.312e-07)\ T_{90}^1 - (9.835e-07)$

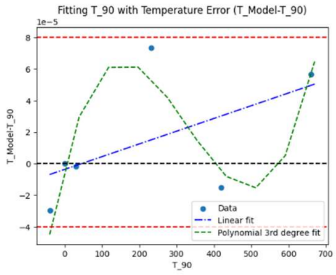


*Figure 16. Fitting curve of error propagations against $T_{90}$*

## CONCLUSION

Strong, effective and accurate pythonic algorithm has buil t that used in calibration of PRT with limited error impact on measurement uncertainty. The interpolation error that a rises when converting resistance values to temperature usi ng ITS-90 equation has validated within the model.
However, discrepancies in temperature values obtained th at can impact measurement reliability has handled into the uncertainty budget. Maximum difference between mode a nd ITS-90 was found to be 173 µK and minimum one wa s equivalent to 3 µK. Conversion constants from the mod el improve the uncertainty from 0.1 mK to 0.06 mK. The model is statistically significant with p-value = 0.999999

## REFERENCES


1. H. Preston-Thomas, "The International Temperature Scale of 1990 (ITS-90)," Metrologia , vol. 27, pp. 3–10, 1990.
2. F. M. Patan and Alper, "The Effect of the Conversion Coefficients of Platinum-Based Resistance Thermometers on the Uncertainty Estimation," International Journal of Thermodynamics, 2023. [Online]. Available: http://doi.org/10.5541/ijot.1220322
3. A. El Matarawy, "Comparison of the realization of water triple point metallic cell through its preparation techniques in new modified adiabatic calorimeter at NIS-Egypt," J. Therm. Anal. Calorim. , 2019. [Online]. Available: https://doi.org/10.1007/s10973-018-7804-8
4. E. C. Jeong and J. Hong Sang, "Machine learning-based virtual metrology on film thickness in amorphous carbon layer deposition process," 2012. [Online]. Available: https://doi.org/10.1016/j.measen.2021.100046
5. S. Rab et al., "Digital Avatar of Metrology," MAPAN, 2023. [Online]. Available: https://doi.org/10.1007/s12647-023-00641-1
6. J. Zhao et al., "A Method for Measuring Tube Metal Temperature of Ethylene Cracking Furnace Tubes Based on Machine Learning and Neural Network," IEEE Access, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2950419
7. T. Ren et al., "Machine learning applied to retrieval of temperature and concentration distributions from infrared emission measurements," Applied Energy , 2019. [Online]. Available: https://doi.org/10.1016/j.apenergy.2019.113448
8. M. W et al., "A novel approach to using artificial intelligence in coordinate metrology including nano scale," Measurement , 2023. [Online]. Available: https://doi.org/10.1016/j.measurement.2023.113051
9. S. E et al., "Metrology for the digital age," Measurement: Sensors , 2021. [Online]. Available: https://doi.org/10.1016/j.measen.2021.100232
10. T. D et al., "Uncertainty-aware data pipeline of calibrated MEMS sensors used for machine learning," Measurement: Sensors , 2022. [Online]. Available: https://doi.org/10.1016/j.measen.2022.100376
11. A. El-Matarawy, "New temperature controlling mechanism for thermal metrology regulation at NIS-Egypt," J. Therm. Anal. Calorim. , 2017. [Online]. Available: https://doi.org/10.1007/s10973-017-6340-2
12. A. El Matarawy and Eman El Dien, "Precise temperature controlling algorithm for metrological adiabatic calorimeters based on proportional integration (α) thermal energy," Journal of Thermal Analysis and Calorimetry. [Online]. Available: https://doi.org/10.1007/s10973-021-10806-2
13. Jupyter Documentation, 2023. [Online]. Available: https://docs.jupyter.org/en/latest/
14. NumPy Documentation, 2023. [Online]. Available: https://numpy.org/news/
15. Pandas Documentation, 2023. [Online]. Available: https://pandas.pydata.org/docs/
16. Matplotlib Documentation, 2023. [Online]. Available: https://matplotlib.org/
17. Plotly Documentation, 2023. [Online]. Available: https://plotly.com/
18. SciPy Documentation, 2023. [Online]. Available: https://scipy.org/
19. Seaborn Documentation, 2023. [Online]. Available: https://seaborn.pydata.org/
20. statsmodels Documentation , 2023. [Online]. Available: https://www.statsmodels.org/stable/index.html
21. *scikit-learn Documentation, 2023. [Online]. Available: https://scikit-learn.org/stable/*
22. *P. Rosenkranz, "Uncertainty Propagation for Platinum Resistance Thermometers Calibrated According to ITS-90," Int. J. Thermophys , vol. 32, pp. 106-119, 2011.*