Procedimiento de calibración DSC

Frente a la necesidad de comenzar un nuevo proceso de calibración en el calorímetro DSC SETARAM, Labsys evo, se decide realizar el procedimiento elaborado en 2013 en un Notebook con la idea de introducir mejoras al mismo, si es posible. Comenzaremos repdroduciendo el modelo anterior utilizando la biblioteca de Python TAL LAL LAL, para posteriormente compararla con los resolutados obtenidos en MATLAB.

Corrección en temperatura

Para la calibración en temperatura se tomaron en cuenta simultáneamente los valores medidos de temperaturas de fusión y las velocidades de calentamiento utilizadas. Se construyó un sistema de ecuaciones lineales usando dichos valores medidos y velocidades seleccionadas como coeficientes conocidos, y los coeficientes de corrección como incógnitas que satisfacen las siguientes ecuaciones:

dT=A0+A1.Te+A2.V+A3.V^2 (1)

dT=Treal–Te (2)

en donde

A0, A1, A2 y A3 son los coeficientes de corrección, V: velocidad de calentamiento/enfriamiento, Te: temperatura medida, Treal: temperatura aceptada en literatura

La expresión dada por la (ec. 1), especializada en los diferentes valores de velocidad de calentamiento utilizados y en las distintas temperaturas de inicio de transformación determinadas, da lugar a un sistema de ecuaciones que posee un número significativamente mayor de ecuaciones que de incógnitas. Esto es, un sistema de ecuaciones sobredeterminado. Estos sistemas no tienen solución; sin embargo, puede encontrarse una pseudosolución. Se optó por resolver el sistema de forma aproximada utilizando los métodos de mínimos cuadrados ordinarios y pseudoinversa de Moore-Penrose.

Modelos de ajuste lineal

un poco de modelos de ajuste lineal BLAH BLAH BLAH

Procedimiento en Python

------> TESTO AQUI Importamos las variables desde una hoja de datos de Excel para luego procesarlas.

In [1]:
```
# Dependencies import
import numpy as np
import pandas as pd

# Reading variables from the sheet
df = pd.read_excel('resultados_v1.xlsx', sheet_name='dT', header = 1)

# Checking the header imported
print(df.head(5))

# Generating NumPy Arrays
Te = df["Te-A"].to_numpy()
Treal = df["Treal-A"].to_numpy()
V = df.V.to_numpy()
dT = Treal - Te
T = np.array([np.ones(dT.shape), Te, V, V ** 2])
```

```
  Material     V     Te-A  Te-B  Treal-A  Treal-B   dT-A  dT-B
0       In   1.0  153.826   NaN  156.599      NaN  2.773   NaN
1       In   5.0  153.829   NaN  156.599      NaN  2.770   NaN
2       In  10.0  153.669   NaN  156.599      NaN  2.930   NaN
3       In  20.0  153.137   NaN  156.599      NaN  3.462   NaN
4       In  50.0  152.623   NaN  156.599      NaN  3.976   NaN
```

Pseudoinversa Moore Penrose

In [2]:
```
# Penrose Matrix with NumPy array
p = np.linalg.pinv(T)

# Check dimensions
print(type(p), np.shape(p), np.size(p))
print(type(dT), np.shape(dT), np.size(dT))
```

```
<class 'numpy.ndarray'> (62, 4) 248
<class 'numpy.ndarray'> (62,) 62
```

In [3]:
```
# Transpose Penrose Matrix for array multiplication
Tp = np.transpose(p)
print(type(Tp), np.shape(Tp), np.size(Tp))

# Multiplying matrix with NumPy array
A = np.matmul(Tp, dT)

# Show the coefficients
print('Coefficients are:')
for i in range(np.size(A)):
    print(f'A{i} = {A[i]:.2e}')
```

```
<class 'numpy.ndarray'> (4, 62) 248
Coefficients are:
A0 = 5.03e+00
A1 = -1.15e-03
A2 = 4.93e-02
A3 = -3.63e-04
```

Mínimos cuadrados

In [4]:
```
# Transpose original Matrix for array multiplication
TT = np.transpose(T)

# Obtaining coefficient through Least Squares
X = np.linalg.lstsq(TT, dT, rcond = -1)

# Show the coefficients
print('Coefficients are:')
for i in range(np.size(X)):
    print(f'A{i} = {X[0][i]:.2e}')
```

```
Coefficients are:
A0 = 5.03e+00
A1 = -1.15e-03
A2 = 4.93e-02
A3 = -3.63e-04
```

```
C:\Users\Pablo\anaconda3\envs\dsc-calibration\lib\site-packages\numpy\core\fromn
umeric.py:3208: VisibleDeprecationWarning: Creating an ndarray from ragged neste
d sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with differ
ent lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  return asarray(a).size
```

Corrección en calor (Q)

Para la calibración en calor se consideró la constante de sensibilidad K, que define el factor de proporcionalidad entre el área del pico medida, entre la curva y la linea de base, y el calor asociado a la transformación que ésta describe.

K = (S/H).(1/m) (3)

K=A0+A1.Te+A2.Te^2+A3.Te^3+A4.Te^4 (4)

siendo A0, A1, A2, A3 y A4 son los coeficientes de corrección, Te: temperatura medida, m: masa, H: calor de fusión, S: área de pico (µV.s)

En este caso, los valores experimentales a considerar como coeficientes conocidos son Te y S. Se aplicó la misma metodología que para corrección en temperatura y se tuvieron en cuenta las mismas consideraciones anteriormente mencionadas.

In [5]:
```
# Reading variables from the sheet
df = pd.read_excel('resultados_v1.xlsx', sheet_name='K', header = 1)

# Checking the header imported
print(df.head(5))

# Generating NumPy Arrays
Te = df["Te-A"].to_numpy()
H = df["H-A"].to_numpy()
S = df["S-A"].to_numpy()
m = df.m.to_numpy()
K = (S/H)*(1/m)
B = np.array([np.ones(Te.shape), Te, Te**2, Te**3, Te**4])
```

```
  Material     V     Te-A  Te-B  Treal-A  Treal-B      S-A  S-B   H-A   H-B  \
0       In   1.0  153.826   NaN  156.599      NaN  718.830  NaN  28.5  28.5
1       In   5.0  153.829   NaN  156.599      NaN  690.485  NaN  28.5  28.5
2       In  10.0  153.669   NaN  156.599      NaN  672.261  NaN  28.5  28.5
3       In  20.0  153.137   NaN  156.599      NaN  657.725  NaN  28.5  28.5
4       In  50.0  152.623   NaN  156.599      NaN  527.595  NaN  28.5  28.5

       m
0  79.47
1  79.47
2  79.47
3  79.47
4  79.47
```

In [6]:
```
print(B.shape)
```

```
(5, 62)
```

Pseudoinversa Moore Penrose

In [7]:
```
# Penrose Matrix with NumPy array
pB = np.linalg.pinv(B)

# Check dimensions
print(type(pB), np.shape(pB), np.size(pB))
print(type(K), np.shape(K), np.size(K))
```

```
<class 'numpy.ndarray'> (62, 5) 310
<class 'numpy.ndarray'> (62,) 62
```

In [8]:
```
# Transpose Penrose Matrix for array multiplication
TpB = np.transpose(pB)
print(type(TpB), np.shape(TpB), np.size(TpB))

# Multiplying matrix with NumPy array
C = np.matmul(TpB, K)

# Show the coefficients
print('Coefficients are:')
for i in range(B.shape[0]):
    print(f'C{i} = {C[i]:.4e}')
```

```
<class 'numpy.ndarray'> (5, 62) 310
Coefficients are:
C0 = 2.8813e-01
C1 = 6.7788e-05
C2 = -7.4088e-07
C3 = 9.5213e-10
C4 = -3.5929e-13
```

Mínimos cuadrados

In [9]:
```
# Transpose original Matrix for array multiplication
TB = np.transpose(B)

# Obtaining coefficient through Least Squares
D = np.linalg.lstsq(TB, K, rcond = -1)

# Show the coefficients
print('Coefficients are:')
for u in range(B.shape[0]):
    print(f'D{u} = {D[0][u]:.4e}')
```

```
Coefficients are:
D0 = 2.8813e-01
D1 = 6.7786e-05
D2 = -7.4087e-07
D3 = 9.5212e-10
D4 = -3.5928e-13
```

Agrego límites teóricos

In [10]:
```
Te = np.append(Te,[-273.0],axis= 0)
Te = np.append(Te,[1600],axis= 0)
K = np.append(K,[0.0],axis= 0)
K = np.append(K,[0.1],axis= 0)

B = np.array([np.ones(Te.shape), Te, Te**2, Te**3, Te**4])
```

In [11]:
```
np.shape(Te)
```

Out[11]:
```
(64,)
```

Pseudoinversa Moore Penrose

In [12]:
```
# Penrose Matrix with NumPy array
pB = np.linalg.pinv(B)

# Check dimensions
print(type(pB), np.shape(pB), np.size(pB))
print(type(K), np.shape(K), np.size(K))
```

```
<class 'numpy.ndarray'> (64, 5) 320
<class 'numpy.ndarray'> (64,) 64
```

In [13]:
```
# Transpose Penrose Matrix for array multiplication
TpB = np.transpose(pB)
print(type(TpB), np.shape(TpB), np.size(TpB))

# Multiplying matrix with NumPy array
C = np.matmul(TpB, K)

# Show the coefficients
print('Coefficients are:')
for i in range(B.shape[0]):
    print(f'C{i} = {C[i]:.4e}')
```

```
<class 'numpy.ndarray'> (5, 64) 320
Coefficients are:
C0 = 2.5029e-01
C1 = 3.8456e-04
C2 = -1.4298e-06
C3 = 1.5062e-09
C4 = -5.0743e-13
```

In [ ]:
```
```