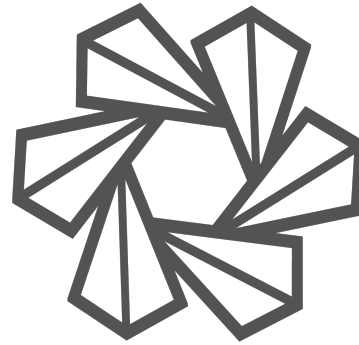
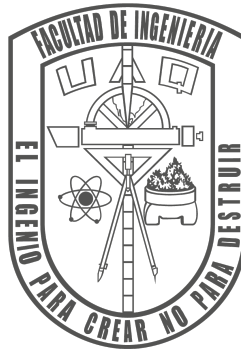
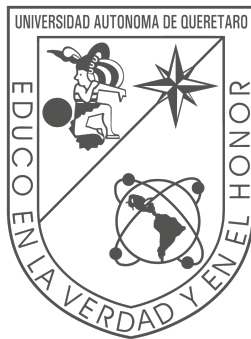


Universidad Autónoma de Querétaro

Facultad de Ingeniería
División de Investigación y Posgrado



Práctica 4 Regresión lineal y polinomial

Maestría en Ciencias en Inteligencia Artificial
Optativa de especialidad IV - Machine Learning

Aldo Cervantes Marquez
Expediente: 262775
Profesor: Dr. Marco Antonio Aceves Fernández

Santiago de Querétaro, Querétaro, México
Semestre 2023-1
21 de Abril de 2023

Índice

1. Objetivo	1
2. Introducción	1
3. Marco Teórico	1
3.1. Métodos de normalización	1
3.1.1. Normalización min-max	1
3.2. Regresión lineal y polinomial	1
3.3. Métodos de regresión	2
3.4. Métricas de error	2
3.5. MSE	3
3.6. RMSE	3
3.7. MAE	3
3.8. RAE	3
3.9. Coeficiente de correlación	3
3.10. Coeficiente de determinación R^2	4
4. Materiales y Métodos	4
4.1. Materiales	4
4.1.1. Base de datos	4
4.1.2. Librerías y entorno de desarrollo	5
4.2. Metodología	5
5. Pseudocódigo	6
6. Resultados	7
6.1. Preprocesamiento y análisis de los datos	7
6.2. Regresión	8
6.3. Temperatura del Refrigerante	9
6.4. Temperatura del devanado del estator	10
6.5. Temperatura del Ambiente	11
6.6. Temperatura de los dientes del estator	12
6.7. Temperatura del Yugo del Estator	13
6.8. Temperatura del Imán Permanente	14
6.9. Par del Motor	15
7. Conclusiones	17
Referencias	18
8. Código Documentado	19

1. Objetivo

Esta práctica tiene como objetivo el de generar una regresión de n grado para poder ajustar conjuntos de puntos para observar tendencias de los mismos y calcular las métricas que permitan observar la concordancia de la regresión utilizada. Aplicado a una base de datos tabular y con serie de tiempo (dependiente de sus observaciones en eventos consecuentes temporales).

2. Introducción

La práctica consiste en obtener una base de datos en serie de tiempo, realizar un análisis de la base de datos, identificando valores atípicos, aplicando métodos de imputación, generación de datos sintéticos y normalización (preprocesamiento de datos). Para posteriormente realizar los métodos de regresión correspondientes para n grado y calcular las métricas que permitan obtener conclusiones de los resultados obtenidos para poder decidir cual regresión se ajusta de mejor manera al comportamiento de los datos.

3. Marco Teórico

Para la realización de la práctica fueron necesarios los siguientes conceptos, los cuales fueron obtenidos principalmente de [1].

3.1. Métodos de normalización

Para este tipo de casos, en datos continuos, se utilizó el siguiente método [2, 3].

3.1.1. Normalización min-max

Esta normalización consiste en realizar un escalamiento entre valores definidos a , b y los datos de serán transformadas a ese rango, donde el valor mínimo estará en a y el máximo en b . Se calcula mediante la siguiente fórmula:

$$v_{norm} = a + \frac{(v_i - v_{min})(b - a)}{v_{max} - v_{min}} \quad (1)$$

3.2. Regresión lineal y polinomial

Consiste en un método para relacionar una variable dependiente y y una variable independiente x . La **regresión lineal** se puede generalizar de la forma:

$$y = f(x) = \beta_0 + \beta_1 x \quad (2)$$

donde:

* β_0 y β_1 : son los coeficientes del modelo lineal (ordenada al origen y pendiente de la recta).

También es posible establecer relaciones no lineales con el fin de obtener un modelo que mejor describa el comportamiento de los datos, se le conoce a esto **regresión polinomial** donde podemos definir al grado n de la función de regresión como:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n \quad (3)$$

Generalmente se utilizan polinomios no mayores a cuarto y quinto grado ($n = 4, 5$).

De manera visual e intuitiva se puede observar las funciones de regresión de la siguiente manera (véase Figura 1).

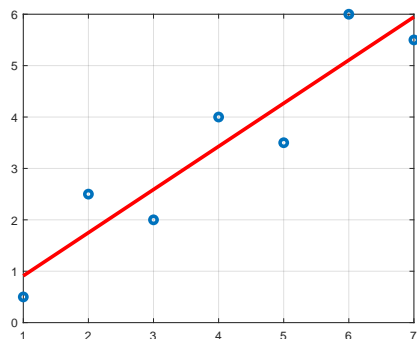


Figura 1: Regresión lineal

3.3. Métodos de regresión

Existen distintos métodos para obtener los coeficientes de la función de regresión, pero en este caso se centrarán en el método de mínimos cuadrados [4], el cual presenta una minimización del error cuadrático. Para obtener los coeficientes de la función de regresión se debe resolver un sistema de ecuaciones que tiene la siguiente estructura.

$$\begin{aligned} n_d \beta_0 + (\sum x_i) \beta_1 + (\sum x_i^2) \beta_2 + \cdots + (\sum x_i^n) \beta_n &= \sum y_i \\ (\sum x_i) \beta_0 + (\sum x_i^2) \beta_1 + (\sum x_i^3) \beta_2 + \cdots + (\sum x_i^{n+1}) \beta_n &= \sum x_i y_i \\ (\sum x_i^2) \beta_0 + (\sum x_i^3) \beta_1 + (\sum x_i^4) \beta_2 + \cdots + (\sum x_i^{n+2}) \beta_n &= \sum x_i^2 y_i \\ &\vdots \\ (\sum x_i^n) \beta_0 + (\sum x_i^{n+1}) \beta_1 + (\sum x_i^{n+2}) \beta_2 + \cdots + (\sum x_i^{2n}) \beta_n &= \sum x_i^n y_i \end{aligned} \quad (4)$$

donde:

* n_d es la longitud de los datos. Todo esto partiendo desde $i = 1$ hasta n_d y n el grado del polinomio.

Esto se realiza mediante la función incrustada de *numpy* `np.polyfit()`

3.4. Métricas de error

Estas permiten conocer con certeza evaluar que tan bueno es el ajuste que fue realizado por el método de regresión [5].

3.5. MSE

Es el error cuadrático medio. El cual permite estimar la cantidad de error (siempre positivo), teniendo como carencia de perder la dirección del error. Su resultado se encuentra en el rango de $[0, \infty)$

$$MSE = \frac{1}{n_d} \sum_{i=1}^{n_d} (\hat{y}_i - y_i)^2 \quad (5)$$

3.6. RMSE

Raíz del error cuadrático medio. Permite de igual manera medir la tasa de error del modelo de regresión. Obteniendo residuales que permiten medir la distancia de los puntos de datos con respecto a la línea de regresión. Definiendo una medida de dispersión de los valores residuales.

$$RMSE = \sqrt{\frac{1}{n_d} \sum_{i=1}^{n_d} (\hat{y}_i - y_i)^2} = \sqrt{MSE} \quad (6)$$

3.7. MAE

Error medio absoluto. Mide la distancia vertical promedio de los puntos evaluados en la función de regresión y las observaciones de la base de datos.

$$MAE = \frac{1}{n_d} \sum_{i=1}^{n_d} |\hat{y}_i - y_i| \quad (7)$$

3.8. RAE

Error relativo Absoluto. Muestra la dispersión entre el valor real y el medido en cada punto.

$$RAE = \frac{\sum_{i=1}^{n_d} |\hat{y}_i - y_i|}{\sum_{i=1}^{n_d} |y_i - \bar{y}|} \quad (8)$$

3.9. Coeficiente de correlación

Se obtiene mediante la covarianza del modelo de regresión con respecto a los valores reales. se define como CC y tiene un rango de $[-1, 1]$ donde 1 significa una correlación positiva perfecta, -1 una correlación negativa perfecta y cuando se tiende a 0 no existe correlación lineal.

$$CC = \frac{S_{PA}}{\sqrt{S_p S_A}} \quad (9)$$

donde:

$$\begin{aligned} S_{PA} &= \left(\frac{\sum_{i=1}^{n_d} (\hat{y}_i - \bar{y})(y_i - \bar{y})}{n_d - 1} \right) \\ S_P &= \frac{\sum_{i=1}^{n_d} (\hat{y}_i - \bar{y})^2}{n_d - 1} \\ S_A &= \frac{\sum_{i=1}^{n_d} (y_i - \bar{y})^2}{n_d - 1} \end{aligned} \quad (10)$$

3.10. Coeficiente de determinación R2

Consiste en el poder de explicación del modelo de regresión y el computo de la suma de los cuadrados. Describe la proporción de varianza de la variable dependiente explicada por el modelo de regresión. Si el modelo de regresión es “perfecto”, SSE es cero, y $R2$ es uno. Si el modelo de regresión es un desastre, SSE es igual a SST , y no se puede explicar ninguna varianza por regresión, además $R2$ es cero.

$$R2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \quad (11)$$

$$\begin{aligned} SST &= \sum (y_i - \bar{y})^2 \\ SSR &= \sum (\hat{y}_i - \bar{y})^2 \\ SSE &= \sum (y_i - \hat{y}_i)^2 \end{aligned} \quad (12)$$

4. Materiales y Métodos

4.1. Materiales

4.1.1. Base de datos

La base de datos elegida para esta práctica fue obtenida de [kaggle](#) [6] de la universidad de Paderborn la cual muestra el comportamiento de un motor síncrono de imán permanente en un banco de pruebas. El motor fue puesto en marcha durante 5 horas y con un periodo de muestreo de 0.5 segundos. Recopilando principalmente temperaturas de las diferentes partes del motor, así como también midiendo su desempeño (torque y corriente) a una velocidad y tensión constante. Por lo que los atributos con los que se trabajará serán:

- Temperatura del refrigerante °C: Consiste en la temperatura dentro del radiador proveniente del sistema de enfriamiento del motor eléctrico. Se define como ‘*coolant*’.
- Temperatura de las bobinas estator °C. Se define como ‘*stator_winding*’.
- Temperatura del ambiente °C: es la temperatura fuera de la armadura del motor. Se define como ‘*ambient*’.
- Temperatura de los dientes del estator °C: temperatura en la parte más cercana al rotor. Se define como ‘*stator*’.

- Temperatura del yugo del estator °C: Es la parte más externa del motor y la que almacena a todo el motor. Se define como *'stator_yoke'*.
- Temperatura del imán permanente (estator) °C: Es la temperatura que se ubica en el imán del estator anclado al yugo. Se define como *'pm'*.
- Par del motor °Nm: Es la cantidad de *'torque'*

Se puede observar de manera gráfica las partes del motor en la Figura 2.

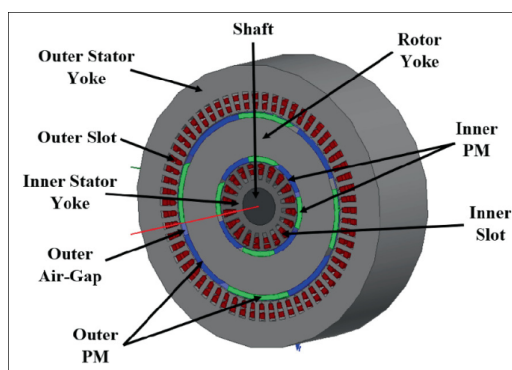


Figura 2: Diagrama de motor PMSM.

Cabe resaltar que esta base de datos consta de más atributos pero no son de tanta importancia como lo son las anteriormente mencionadas.

También se debe considerar que se extraerán los datos ocurridos entre las instancias 600 a 1500 (5 a 12.5 minutos). Teniendo un total de 900 instancias y 7 atributos. Esto debido a que se considera un estado aún transitorio del sistema, casi llegando al estado estacionario.

4.1.2. Librerías y entorno de desarrollo

El análisis de los datos se llevará a cabo en el lenguaje de programación Python dentro del entorno de desarrollo de Jupyter Notebook. Ocupando las librerías [matplotlib](#), [seaborn](#), [numpy](#) y [Pandas](#).

4.2. Metodología

La metodología consiste en la recopilación de las bases de datos, aplicar métodos de preprocesamiento de datos, realizar análisis de correlación entre atributos y generar las regresiones correspondientes para finalmente medir sus métricas de evaluación (véase Figura 3).

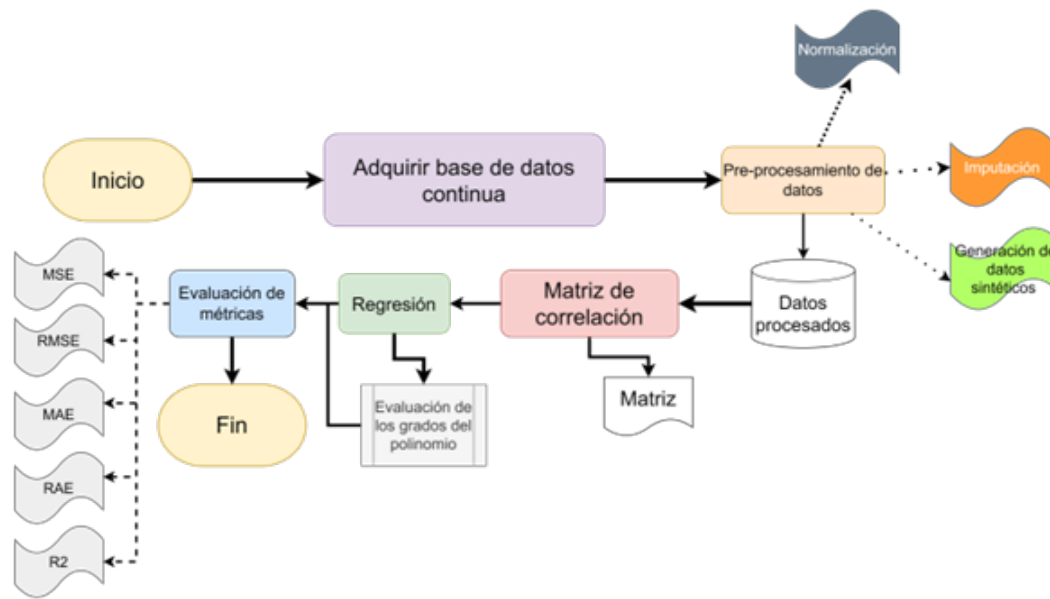


Figura 3: Metodología de la práctica.

5. Pseudocódigo

Algoritmo 1 Pseudocódigo regresión.

```

Inicio
  Class preprocesamiento:
    Normalización
    Imputación
    Generación de datos sintéticos
  Class métricas:
    MSE
    RMSE
    MAE
    RAE
    CC
    R2
  Func regresión(x,y,n)

  DB ← preprocesamiento(DB)
  Para ia en n:
    res(ia) ← regresión(DB(x),DB(y),ia)
    met(ia) ← métricas(res(ia))
Fin
  
```


6. Resultados

6.1. Preprocesamiento y análisis de los datos

Se realizó una normalización *min-max* en los atributos para poder tener todo en las mismas unidades (adimensional), y de esta manera poder tener una mejor comparación entre atributos.

En la Figura 4 se observa la correlación entre las variables, las cuales permiten observar cual es la relación que hay entre cada variable para poder observar alguna tendencia que pueda relacionar cada atributo y obtener conclusiones a partir de ello. Como por ejemplo que el refrigerante, el torque y el ambiente, tienen poca correlación con los demás atributos.

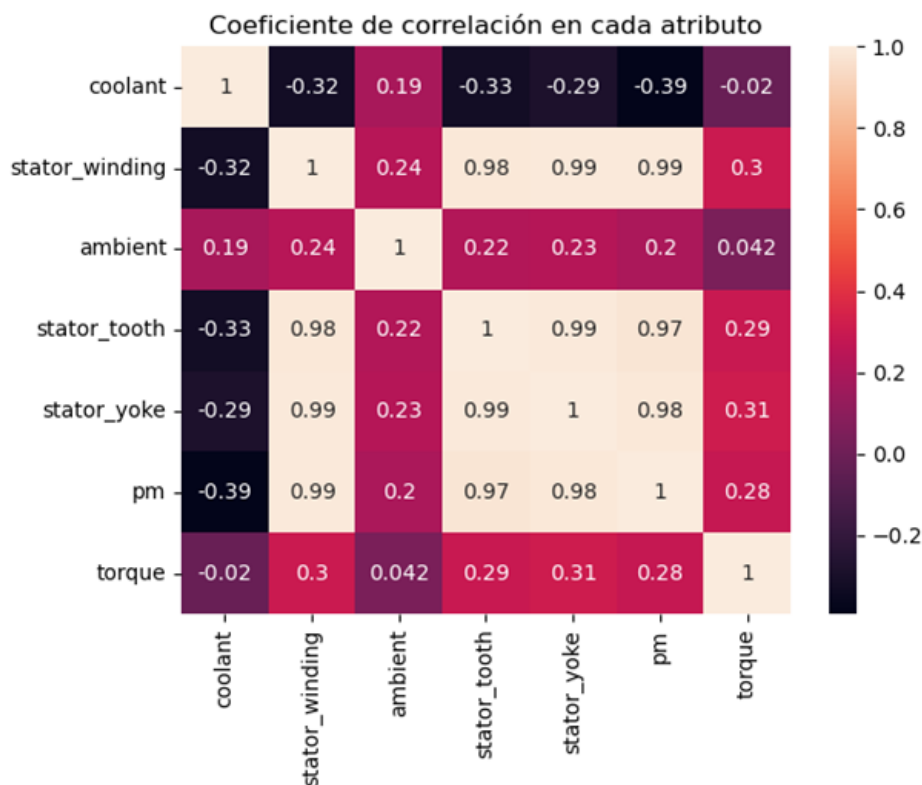


Figura 4: Matriz de correlación entre atributos.

También se realizó una correlación con respecto al tiempo, obteniendo lo siguiente:

Tabla 1: Correlación de los atributos con respecto al tiempo.

$coolant = -0.4585$	$Stator_{winding} = 0.970$	$ambient = 0.1730$
$stator_{tooth} = 0.9437$	$stator_{yoke} = 0.9577$	$pm = 0.9925$
$torque = 0.2611$		

Esto significa que tienen cierta correlación con el tiempo que va pasando (ascendente), siendo el ambiente y el torque los menos correlacionados con el tiempo.

Finalmente se realizó un gráfico de caja para poder observar la dispersión e identificar valores atípicos (véase Figura 5).

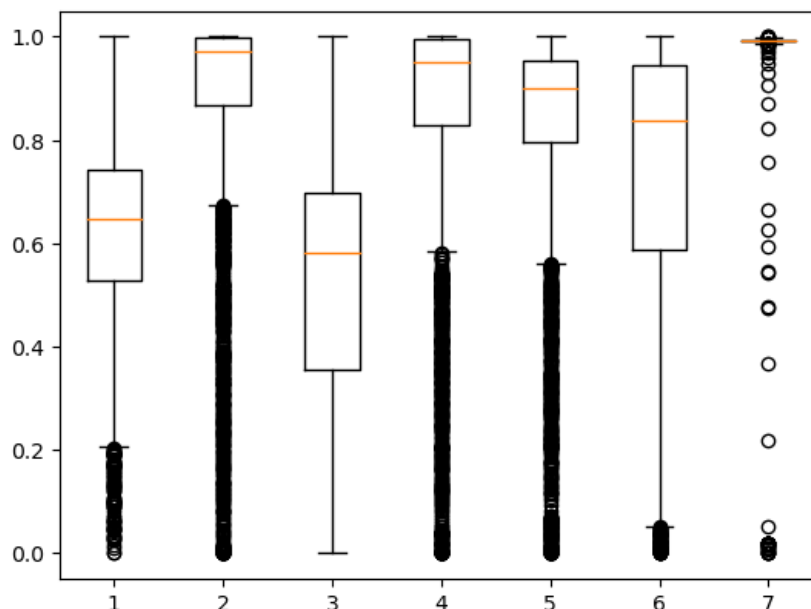


Figura 5: diagrama de caja de los atributos.

donde:

1. *coolant*
2. *stator_winding*
3. *ambient*
4. *stator_tooth*
5. *stator_yoke*
6. *pm*
7. *torque*

Se observa que hay bastantes valores atípicos, esto se debe principalmente a la sensibilidad de los sensores, lo que puede permitir algunos errores en la medición o picos que generen ruido en la señal.

6.2. Regresión

Se realizó regresión polinomial en cada una de los atributos con respecto al tiempo. Obteniendo los siguientes resultados:

6.3. Temperatura del Refrigerante

Se realizó una regresión hasta el grado 6 (véase la Figura 6) con su cambio en las métricas (Figura 7), pero se exploró hasta el grado 10 (véase la Figura 8).

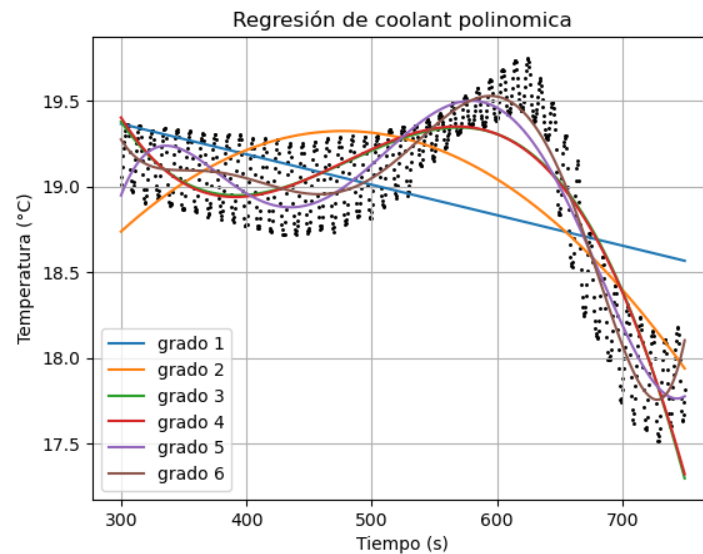


Figura 6: Regresión hasta grado 6 de la temperatura del refrigerante.

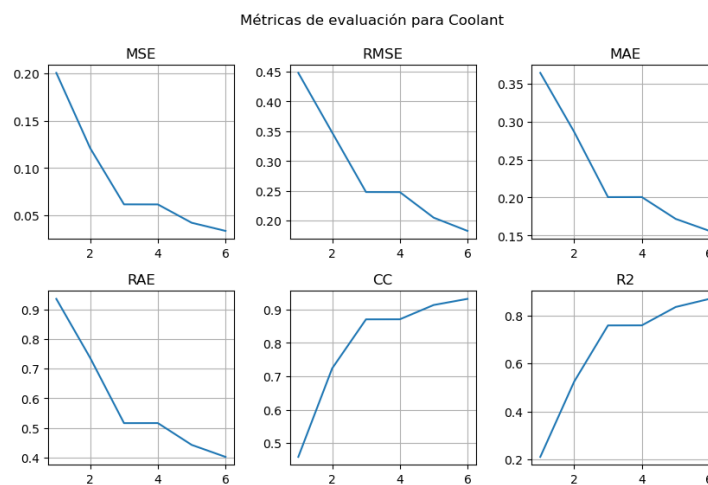


Figura 7: Métricas hasta grado 6 de la temperatura del refrigerante.

Se observa que hay un estancamiento en el grado 3 y 4 pero después se sigue reduciendo, por lo que se puede decir que en los grados siguientes no habrá una mejoría más significativa.

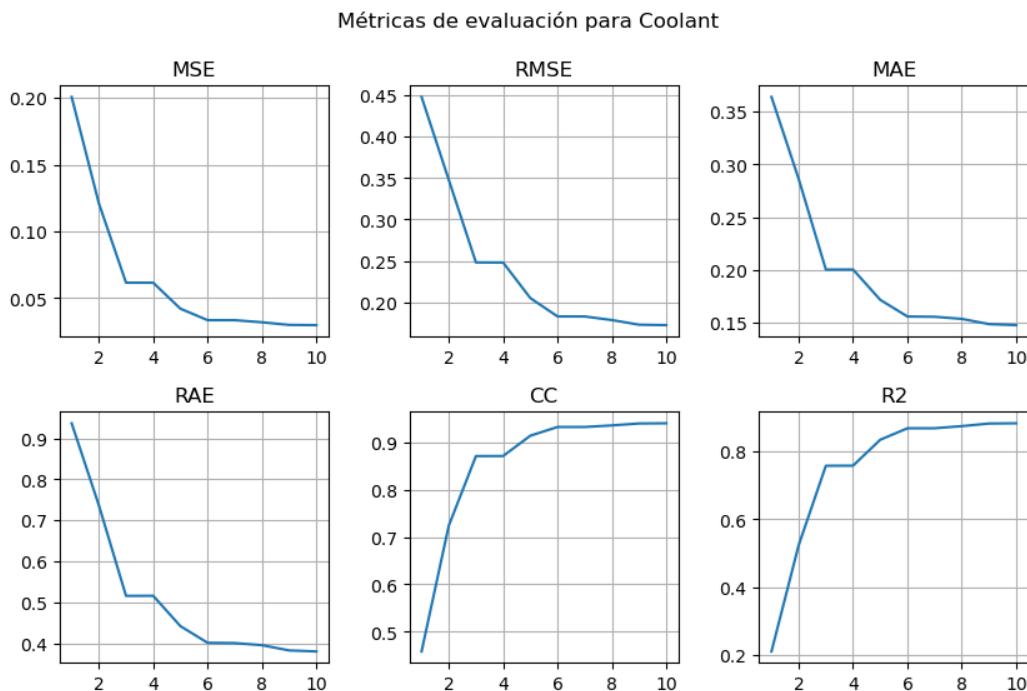


Figura 8: Regresión hasta grado 6 de la temperatura del refrigerante.

Como fue dicho con anterioridad después del orden 5 o 6 no hubo una mejoría significativa por lo que se puede mantener en este caso así.

Se observó que este sistema tenía un comportamiento oscilante a lo largo del tiempo, esto debido al principio de funcionamiento del sistema de refrigeración.

6.4. Temperatura del devanado del estator

Como se observa en la Figura 9 a partir del segundo grado, se tiene una regresión bastante aceptable a simple vista. Esto debido a que se encuentra en un estado transitorio, es posible observar como va incrementandose la temperatura del componente.

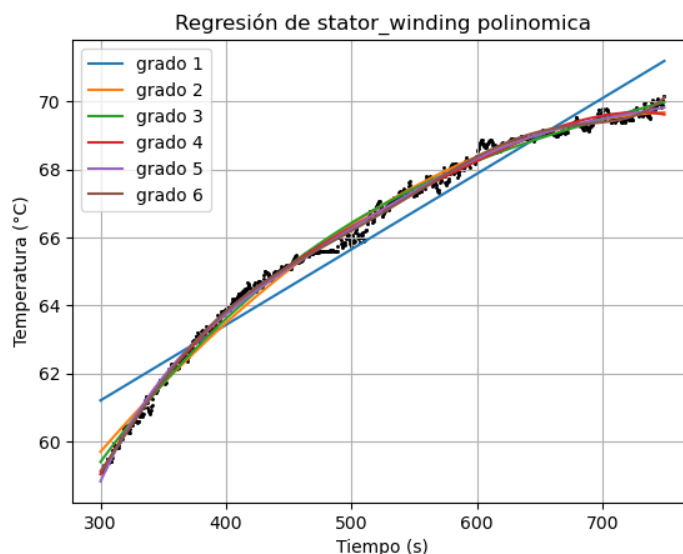


Figura 9: Regresión hasta grado 6 de la temperatura del devanado del estator.

Comprobándose en la Figura 10 que a partir del segundo grado no existe un cambio significativo en las métricas.

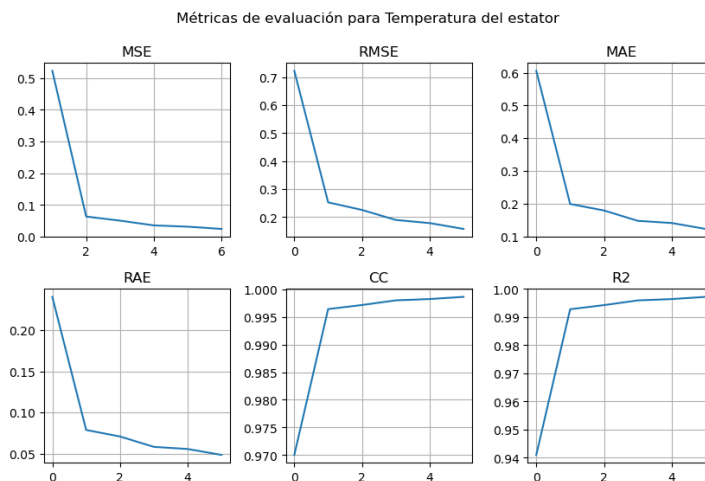


Figura 10: Métricas hasta grado 6 de la temperatura del devanado del estator.

6.5. Temperatura del Ambiente

Se observa que la temperatura del ambiente es algo inestable (véase Figura 11), debido a la cantidad de valores atípicos que se tienen, tomando en cuenta ello, las métricas seguirán mejorando conforme se aumente el grado (véase Figura 12), hasta llegar a alguno bastante alto.

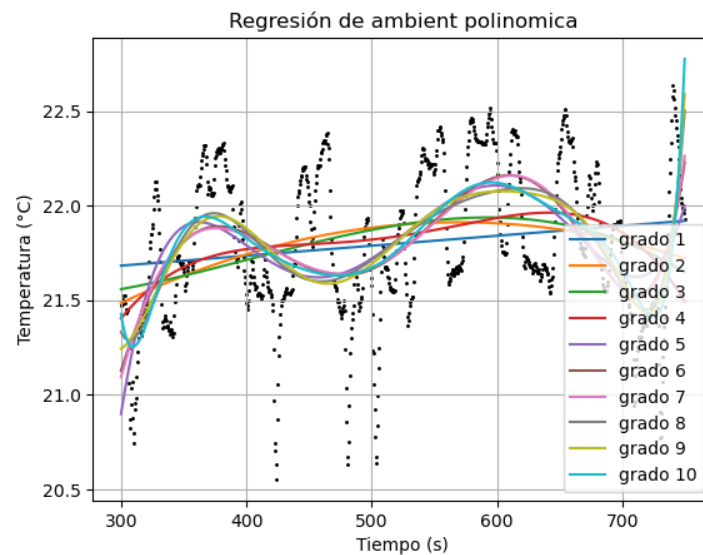


Figura 11: Regresión hasta grado 10 de la temperatura del ambiente.

Desafortunadamente eso no significará que el modelo sea mucho mejor, pues se tiene una función de regresión mucho más compleja y probablemente haya ruido en la señal de los sensores.

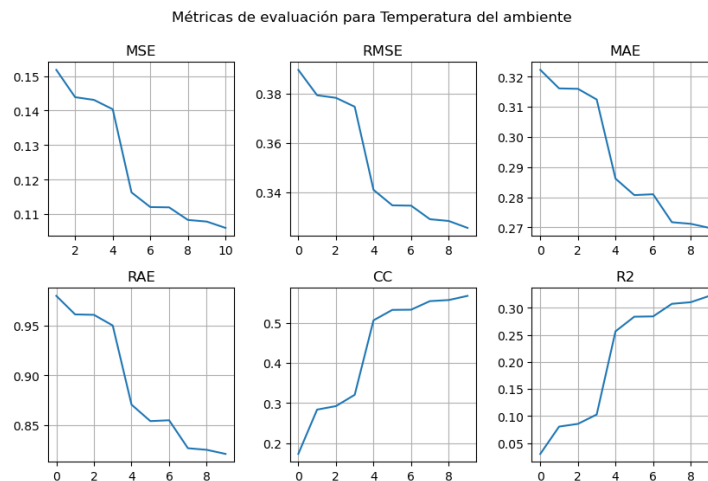


Figura 12: Métricas hasta grado 10 de la temperatura del ambiente.

6.6. Temperatura de los dientes del estator

Debido a que es una parte interna del motor, tendrá un calentamiento en su componente, observándose de la siguiente manera (véase Figura 13):

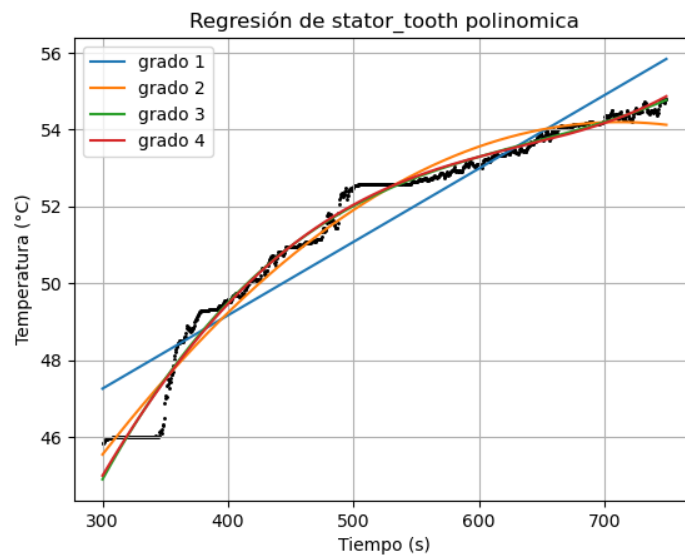


Figura 13: Regresión hasta grado 4 de la temperatura de los dientes del estator.

Observando las métricas, se puede decir que con un grado 3 se tendrá una buena representación de los datos (véase Figura 14).

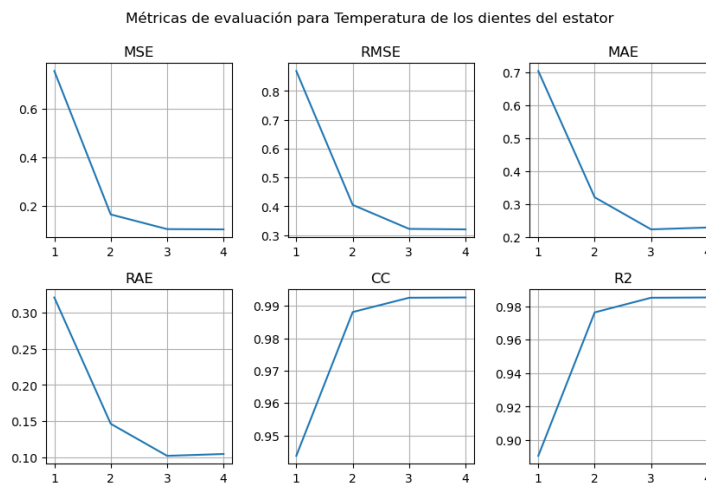


Figura 14: Métricas hasta grado 4 de la temperatura de los dientes del estator.

6.7. Temperatura del Yugo del Estator

Este atributo también tiene un comportamiento creciente a lo largo del tiempo, pues su temperatura se va incrementando y estabilizando (véase Figura 16).

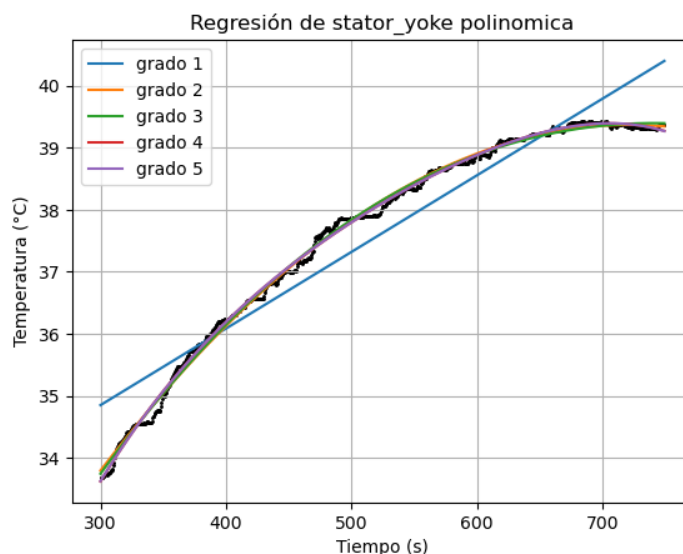


Figura 15: Métricas hasta grado 5 de la temperatura del yugo del estator.

Por otro lado se observa como es que las métricas tienden a estabilizarse a partir del grado 2, por lo que no será necesario agregar un grado mayor para obtener una mejoría significativa.

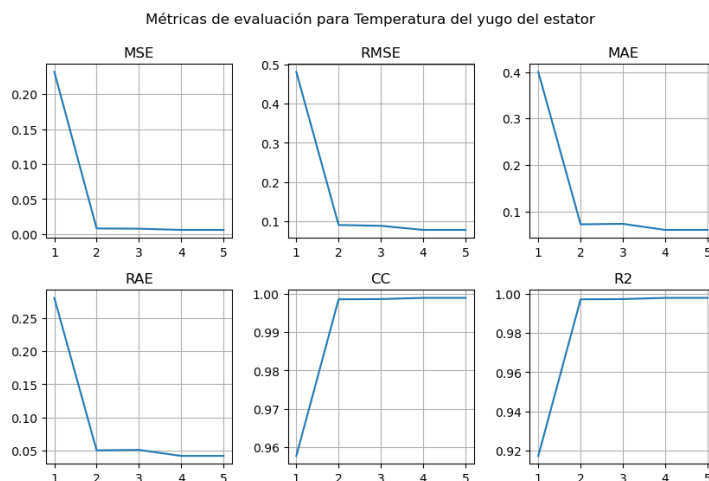


Figura 16: Métricas hasta grado 5 de la temperatura del yugo del estator.

6.8. Temperatura del Imán Permanente

Debido a que es una parte incrustada en el motor (estator), tendrá el mismo comportamiento que el estator, el cual es incrementar su temperatura a lo largo del tiempo, pero en este caso con otro tipo de tasa de cambio, pues es un material con otras características térmicas (véase Figura 17).

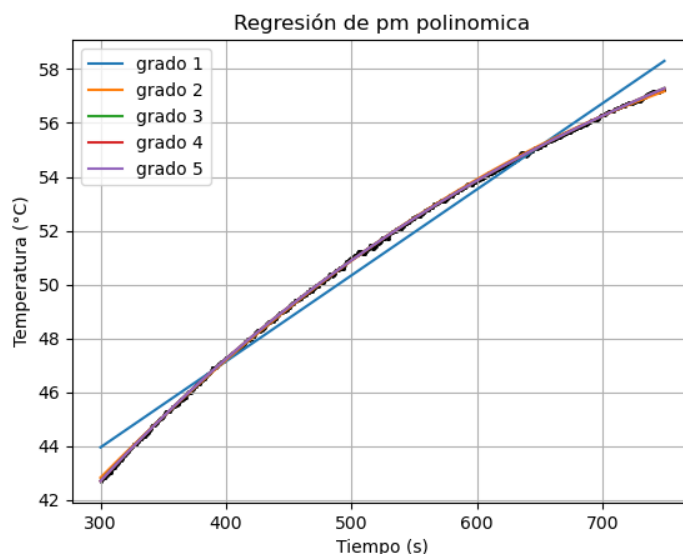


Figura 17: Regresión hasta grado 5 de la temperatura del imán permanente.

Las métricas muestran como a partir del grado 2, es posible tener una regresión bastante buena como se muestra en la Figura 18.

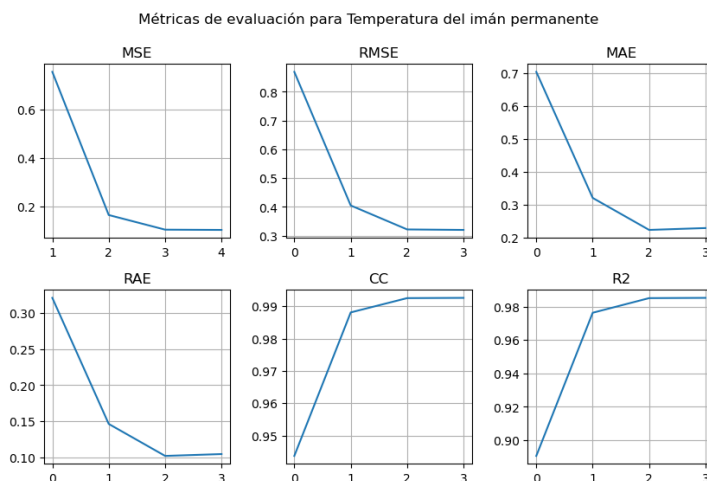


Figura 18: Métricas hasta grado 5 de la temperatura del imán permanente.

6.9. Par del Motor

Este atributo tiene una baja correlación con respecto al tiempo (Figura 19), por lo que es bastante complicado encontrar una función de un orden bajo que pueda representar los puntos de una manera fidedigna, pues será necesario conocer más aspectos de esta.

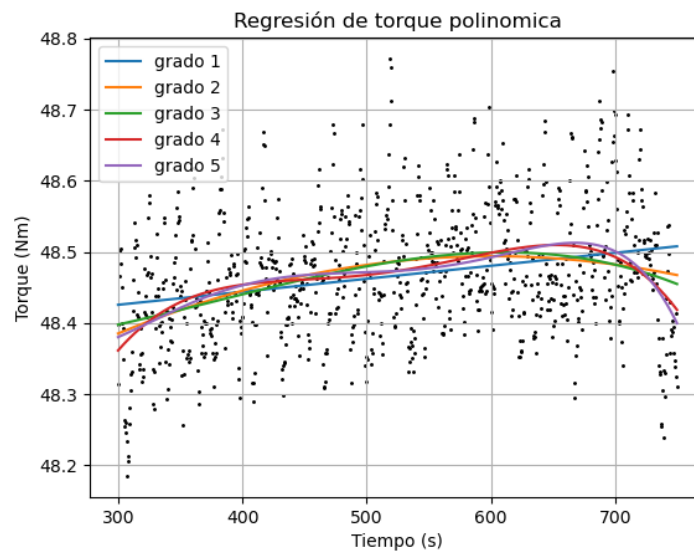


Figura 19: Regresión hasta grado 5 del torque del motor.

Conforme se va incrementando el grado se observa una mejora en las métricas, sin embargo, se puede observar que es muy baja la cantidad de error que se está arreglando. Además de que por ejemplo la métrica MAE tiene una crecida en el grado 3, por lo que eso puede significar que no es muy concordante la curva con los datos (véase Figura 20).

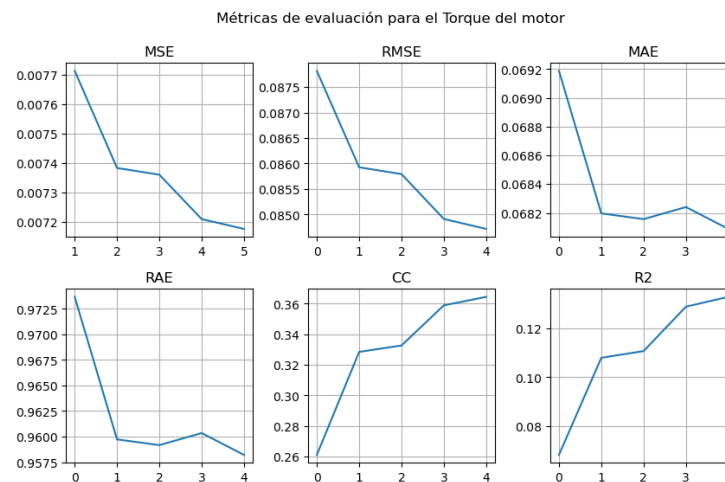


Figura 20: Métricas hasta grado 5 del torque del motor.

[7]

7. Conclusiones

La presente práctica mostró la aplicación de modelos de regresión polinomiales en una base de datos que contiene series de tiempo, esto significa que los eventos serán medidos en instantes de tiempo y tendrán importancia las instancias anteriores con la actual. La base de datos analizada fue la de temperatura en un motor eléctrico PMSM, donde una universidad alemana proporcionó los datos obtenidos por su banco de pruebas, explicando en los metadatos su tiempo de muestreo y un poco sobre sus sensores. Esta base de datos fue preprocesada para poder realizar algunos cálculos de correlación entre las variables y se observaron comportamientos entre los atributos, mostrando lo siguiente:

- Los datos de la temperatura del refrigerante no tenían mucha correlación con los demás datos, pues su sistema de refrigeración consiste en calentar y enfriar el motor de una manera periodica (sinusoidal).
- La temperatura de los componentes del motor tienen mucha relación entre si, pues al encontrarse cerca de la fuente de calor y en contacto entre si, se puede considerar que en un tiempo muy largo, alcanzarán un equilibrio térmico.
- También se observó que la temperatura del ambiente iba incrementando poco a poco pues se intentaba de igual manera llegar al equilibrio térmico pero no era tan sencillo pues interactuaban más factores externos de la
- El segmento del estado (transitorio o estacionario) del sistema, determinará de una manera más completa realizar la regresión.
- Para este tipo de sistemas, se puede realizar una segmentación de los estados del sistema (**como un trabajo a futuro**), pudiendo agregar métodos de agrupamiento para conocer los estados transitorios y estacionarios, y de esta manera identificar los segmentos a los que se les aislará y aplicará regresión, para finalmente unir cada resultado (spline).
- Sería útil agregar más métricas que permitan conocer la dirección del error, como lo es el sesgo, el cual permite conocer si el modelo que se está proponiendo es muy sencillo para el conjunto de datos.
- El torque no se ve afectado por el incremento de temperatura en ese intervalo de tiempo, puesto que al tener relativamente poca variación, es posible tomar como buena alguna de las tendencias mostradas.
- Debido a la alta varianza de las mediciones, es posible que se pueda realizar un suavizamiento de la señal mediante filtros (pasa bajas, pasa altas, etc.) o métodos estadísticos o heurísticos (media móvil, splines, etc.)

Además de que el grado de los polinomios no excedió el 5to, pues se puede volver más complicada la función y podría dejar de explicar las tendencias del sistema por factores externos como valores

atípicos o errores en la medición (incertidumbre, sensibilidad, reproducibilidad, calibración, ruido electromagnético, etc.) pudiendo convertir la regresión en una interpolación.

Finalmente se pudo concluir que con este tipo de estudios de regresión, es posible entender a profundidad la interacción entre los atributos de una base de datos, lo que permitirá que se puedan llegar a conclusiones lógicas sin necesidad de ser un especialista en el tema. como por ejemplo se investigó aparte los efectos de la temperatura con respecto al desempeño del motor (específicamente el torque), encontrando que a pesar de que es un factor que afecta el comportamiento del motor, este durante periodos cortos de tiempo, no se ve afectado de una manera importante [7], lo cual fue observado mediante la tabla de correlación y sus respectivas regresiones.

Referencias

- [1] M. Fernández, *Inteligencia Artificial para Programadores con Prisa*. Amazon Digital Services LLC - KDP Print US, 2021.
- [2] S. Lasse, “Data augmentation for tabular data.” <https://medium.com/analytics-vidhya/data-augmentation-for-tabular-data-f75c94398c3e>, November 2021. (Accessed on 03/16/2023).
- [3] L. Al Shalabi and Z. Shaaban, “Normalization as a preprocessing engine for data mining and the approach of preference matrix,” in *2006 International Conference on Dependability of Computer Systems*, pp. 207–214, 2006.
- [4] C. Steven and C. Raymond, *Métodos numéricos para ingenieros*. MC Grawn Hill, 2007.
- [5] Datascience, “Evaluación por regresión.” <http://datascience.esy.es/wiki/evaluacion-por-regresion/>, Marzo 2018. (Accessed on 04/10/2023).
- [6] W. Kirchgässner, O. Wallscheid, and J. Böcker, “Electric motor temperature,” 2021.
- [7] J. R. Gómez Sarduy, P. R. Viego Felipe, M. A. de Armas Teyra, and M. García Abreu, “Procedimiento para el cálculo de los parámetros de un modelo térmico simplificado del motor asincrónico,” *Ingeniare. Revista chilena de ingeniería*, vol. 19, pp. 122 – 131, 06 2011.

practica4_regresion_aldo_cervantes

April 21, 2023

```
[1]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
C:\Users\aldoa\anaconda3\envs\env2\lib\site-packages\scipy\__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version
of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}")
```

```
[2]: data=pd.read_csv('tempm30min.csv') # 0 a 3600 mediciones
data.columns
```

```
[2]: Index(['Unnamed: 0', 'coolant', 'stator_winding', 'ambient', 'stator_tooth',
          'stator_yoke', 'pm', 'torque'],
          dtype='object')
```

```
[3]: data.isna().sum()
## No hay datos faltantes.
```

```
[3]: Unnamed: 0      0
coolant           0
stator_winding    0
ambient           0
stator_tooth      0
stator_yoke       0
pm                0
torque            0
dtype: int64
```

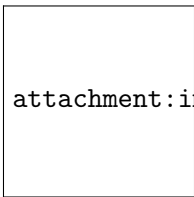
0.1 Significado de la base de datos

Comprende datos obtenidos mediante sensores de un motor sincrónico de imán permanente (PMSM) en un banco de pruebas. Los datos fueron obtenidos en la universidad Paderborn, las mediciones se realizaron cada 0.5 segundos (2 Hz).

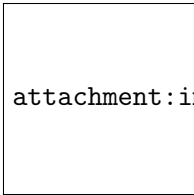
Fue obtenida de [kaggle](#)

Cada atributo significa:

- *u_q*: tensión del motor (V).
- *coolant*: temperatura del refrigerante del motor (°C).
- *stator_winding*: temperatura del estator (°C).
- *u_d*: voltaje del componente d (V).
- *stator_tooth*: temperatura de los dientes del estator (°C).
- *motor_speed*: velocidad angular del motor (R.P.M.)
- *i_d*: corriente del compoente d motor (A).
- *i_q*: Corriente del componente q (A).
- *pm*: temperatura del imán permanente (°C).
- *stator_yoke*: temperatura del yugo del estator (°C).
- *ambient*: temperatura del ambiente (°C).
- *torque*: par del motor (Nm).
- *profile_id* sesion de medicion (identificador).



attachment:image.png



attachment:image-2.png

0.1.1 Consideraciones

Se toma en cuenta un segmento de la prueba total, pues existen muchos estados transitorios que no son de mucho interés, por lo que se tomarán los estados estables con sus respectivas mediciones. **se considera tensión y velocidad constante.**

```
[4]: # seg a min
def mtomin(inicio,fin):
    if inicio>=fin:
        print('error de tiempos')
        return 0

    ini=inicio/2
    fi=fin/2
    print('inicio (seg):', ini)
    print('inicio (min):',(ini)/60)
    print('fin (seg):', fin)
    print('fin (min):',(fi)/60)
    print('total de mediciones:', fin-inicio)
    print('tiempo analizado (min):', (fi/60)-(ini/60))
```

```

inicio=600
fin=1500
mtomin(inicio,fin)
tiempo=np.arange(inicio,fin)/2 #tiempo reiniciado
print(tiempo.shape)

```

```

inicio (seg): 300.0
inicio (min): 5.0
fin (seg): 1500
fin (min): 12.5
total de mediciones: 900
tiempo analizado (min): 7.5
(900,)

```

```

[5]: motor_db=data.copy()
motor_db=motor_db.drop(['Unnamed: 0'],axis=1)
#motor_db.rename(columns={'Unnamed: 0':'tiempo'},inplace=True)
#motor_db['tiempo']=motor_db['tiempo'].replace(tiempo)

```

1 Normalización de valores

Se realizará una normalización min-max entre $a = 0$ y $b = 1$, con el fin de evitar trabajar con números demasiado grandes.

$$v_{norm} = a + \frac{(v_i - v_{min})(b - a)}{v_{max} - v_{min}}$$

```

[6]: def norm_min_max(x,a,b): # Para todo el dataframe
    l=list(x.columns)
    v_max=0
    v_min=0
    res=pd.DataFrame()
    for val in l:
        v_max=x[val].max()
        v_min=x[val].min()
        r_dt=v_max-v_min
        r_norm=b-a
        d=x[val]-v_min
        dpct=d/r_dt
        dnorm=r_norm*dpct
        data=a+dnorm
        aa=pd.DataFrame(data,columns=[val])
        res[val]=data
    return res

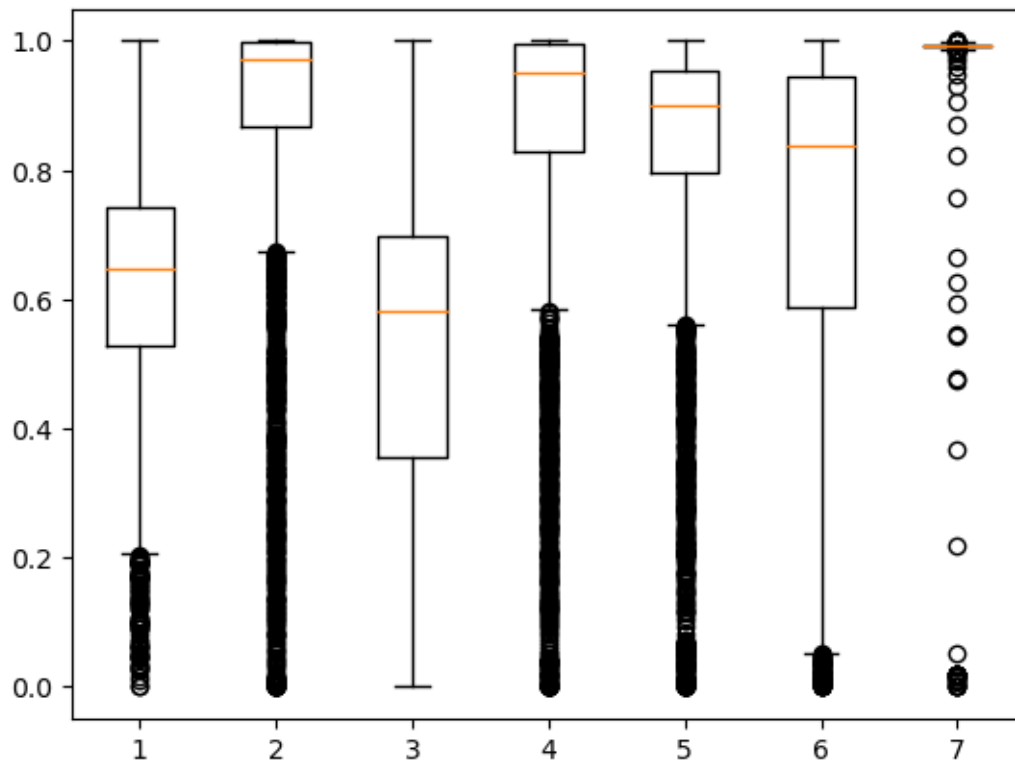
```

```
[7]: motor_db_norm=norm_min_max(motor_db,0,1)
print(motor_db_norm.dtypes)
```

```
coolant          float64
stator_winding    float64
ambient          float64
stator_tooth      float64
stator_yoke       float64
pm               float64
torque           float64
dtype: object
```

```
[8]: ## BOXPLOTS.
bplot=plt.boxplot(motor_db_norm)
print(motor_db_norm.columns)
```

```
Index(['coolant', 'stator_winding', 'ambient', 'stator_tooth', 'stator_yoke',
      'pm', 'torque'],
      dtype='object')
```



2 Regresión

Es un método para estudiar la relación entre una variable dependiente y y una variable independiente x . La regresión lineal se puede generalizar mediante una función:

$$y = f(x) = \beta_0 + \beta_1 x$$

donde: * β_0 y β_1 : son los coeficientes del modelo lineal (ordenada al origen y pendiente de la recta).

También es posible establecer relaciones no lineales con el fin de obtener un modelo que mejor describa el comportamiento de los datos, se le conoce a esto regresión polinomial donde podemos definir al grado n de la función de regresión como:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \dots + \beta_n x^n$$

Generalmente se utilizan polinomios no mayores a cuarto y quinto grado ($n = 4, 5$).

Existen muchos métodos para obtener los coeficientes de la función de regresión, por ejemplo mínimos cuadrados, con su respectiva cuantificación del error llamado *error estándar del estimado* y el coeficiente de correlación del modelo.

De manera general se pueden obtener los coeficientes resolviendo un sistema de ecuaciones lineales con la siguiente forma:

$$\begin{aligned} n_d \beta_0 + (\sum x_i) \beta_1 + (\sum x_i^2) \beta_2 + \dots + (\sum x_i^n) \beta_n &= \sum y_i \\ (\sum x_i) \beta_0 + (\sum x_i^2) \beta_1 + (\sum x_i^3) \beta_2 + \dots + (\sum x_i^{n+1}) \beta_n &= \sum x_i y_i \\ (\sum x_i^2) \beta_0 + (\sum x_i^3) \beta_1 + (\sum x_i^4) \beta_2 + \dots + (\sum x_i^{n+2}) \beta_n &= \sum x_i^2 y_i \\ &\vdots \\ (\sum x_i^n) \beta_0 + (\sum x_i^{n+1}) \beta_1 + (\sum x_i^{n+2}) \beta_2 + \dots + (\sum x_i^{2n}) \beta_n &= \sum x_i^n y_i \end{aligned}$$

donde: * n_d es la longitud de los datos todo esto yendo desde $i = 1$ hasta n_d y n el grado del polinomio.

Por otro lado se tiene la regresión polinomial de la función de `np.polyfit()` de la cual obtiene los coeficientes a partir de la minimización del error cuadrático:

$$\min(E = \sum_{j=0}^k |p(x_j) - y_j|^2)$$

2.0.1 Regresión Lineal Múltiple

Se obtiene mediante técnicas de machine Learning [regresión múltiple](#) y [regresión múltiple IBM](#).

```
[9]: # Regresión lineal con mínimos cuadrados
def polinomyal(x,y,n):
    A=np.zeros((n+1,n+1))
    prom_y=y.mean()
    b=np.zeros((n+1,1))
    contador=0
```

```

for col in range(0,n+1):

    for ren in range(0,n+1):
        if col==0 and ren==0:
            A[ren,col]=len(x)
        else:
            A[ren,col]=sum(x**(contador+ren))

    contador+=1

for o in range(n+1):
    b[o,0]=sum(x**(o)*y)

res=np.linalg.inv(A).dot(b)
print(x.shape)
s_t=0
s_r=0
ev_y=[]
eval_temp=0
for ll in range(len(x)):
    for sust in range(len(res)):
        eval_temp=eval_temp+(res[sust,0]*(x[ll]**(sust)))

    ev_y.append(eval_temp)
    eval_temp=0
    s_r=s_r+(y[ll]-ev_y[-1])**2
    s_t=s_t+(y[ll]-prom_y)**2
r=np.sqrt((s_t-s_r)/(s_t))
print('Resultado:', res)
print('Coeficiente de determinación: ',r)
plt.scatter(x,y)
plt.plot(x,ev_y, color='red')
plt.grid()
plt.title('Regresión polinomial')
return [res,r]

```

3 Métricas de error

Obtenidas del libro y de [métricas de error en regresión](#)

3.1 MSE

Error cuadrático medio (Mean Squared Error) El método más común para predecir el error y la raíz también.

$$MSE = \frac{1}{n_d} \sum_{i=1}^{n_d} (\hat{y}_i - y_i)^2$$

donde \hat{y}_i es la predicción en la posición i , y_i es el valor observado y n_d la longitud de los datos

3.2 RMSE

Raíz del Error cuadrático medio (Root Mean Squared Error)

$$RMSE = \sqrt{\frac{1}{n_d} \sum_{i=1}^{n_d} (\hat{y}_i - y_i)^2}$$

En un rango de $[0, \infty)$

3.3 MAE

Error Medio Absoluto (Mean Absolute Error)

$$MAE = \frac{1}{n_d} \sum_{i=1}^{n_d} |\hat{y}_i - y_i|$$

RAE Error relativo Absoluto (Relative Absolute Error)

$$RAE = \frac{\sum_{i=1}^{n_d} |\hat{y}_i - y_i|}{\sum_{i=1}^{n_d} |y_i - \bar{y}|}$$

Donde \bar{y} es la media de los datos reales observados

3.4 CC

Coefficiente de correlación (Correlation Coefficient)

$$CC = \frac{S_{PA}}{\sqrt{S_P S_A}}$$

donde:

$$S_{PA} = \left(\frac{\sum_{i=1}^{n_d} (\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{n_d - 1} \right)$$

$$S_P = \frac{\sum_{i=1}^{n_d} (\hat{y}_i - \bar{\hat{y}})^2}{n_d - 1}$$

$$S_A = \frac{\sum_{i=1}^{n_d} (y_i - \bar{y})^2}{n_d - 1}$$

Donde $\bar{\hat{y}}$ es la media de los valores predichos por el modelo de regresión ## R2
Coeficiente de determinación R^2 resume el poder de explicación del modelo de regresión y el cómputo de la suma de los cuadrados

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

donde: *Sum of Squares Total*

$$SST = \sum (y_i - \bar{y})^2$$

Sum of Squares Regression

$$SSR = \sum (\hat{y}_i - \bar{\hat{y}})^2$$

Sum of Squares Error

$$SSE = \sum (y - \hat{y})^2$$

describe la proporción de varianza de la variable dependiente explicada por el modelo de regresión. Si el modelo de regresión es “perfecto”, *SSE* es cero, y *R2* es uno. Si el modelo de regresión es un desastre, *SSE* es igual a *SST*, y no se puede explicar ninguna varianza por regresión, además *R2* es cero.

```
[10]: def mse(y,yp):
    nd=len(y)
    temp=0
    #y=y.reset_index(drop='True')
    #yp=yp.reset_index(drop='True')
    for a in range(nd):
        temp=temp+(yp[a]-y[a])**2
    temp/=nd
    return temp

def rmse(y,yp):
    ms=np.sqrt(mse(y,yp))
    return ms

def mae(y,yp):
    nd=len(y)
    temp=0
    #y=y.reset_index(drop='True')
    #yp=yp.reset_index(drop='True')
    for a in range(nd):
        temp=temp+abs(yp[a]-y[a])
    temp/=nd
    return temp

def rae(y,yp):
    nd=len(y)
    ym=y.mean()
    ypm=yp.mean()
    temp=0
    temp1=0
    temp2=0
    #y=y.reset_index(drop='True')
    #yp=yp.reset_index(drop='True')
    for a in range(nd):
        temp1=temp1+abs(yp[a]-y[a])
```

```

        temp2=temp2+abs(y[a]-ym)
    return temp1/temp2

def cc(y,yp):
    nd=len(y)
    ym=y.mean()
    ypm=yp.mean()
    spa=0
    sp=0
    sa=0
    #y=y.reset_index(drop='True')
    #yp=yp.reset_index(drop='True')
    for a in range(nd):
        spa=spa+((yp[a]-ypm)*(y[a]-ym))
        sp=sp+(yp[a]-ypm)**2
        sa=sa+(y[a]-ym)**2
    spa/=(nd-1)
    sp/=(nd-1)
    sa/=(nd-1)
    return spa/(np.sqrt(sp*sa))

def r2(y,yp):
    nd=len(y)
    sst=0
    sse=0
    ym=y.mean()
    # ypm=yp.mean()
    #y=y.reset_index(drop='True')
    #yp=yp.reset_index(drop='True')
    for a in range(nd):
        sse=sse+(y[a]-yp[a])**2
        sst=sst+(y[a]-ym)**2
    return ((sst-sse)/sst)

print(motor_db_norm.columns)

    # s_r=s_r+(y[ll]-ev_y[-1])**2
    # s_t=s_t+(y[ll]-prom_y)**2
    #r=np.sqrt((s_t-s_r)/(s_t))

```

```

Index(['coolant', 'stator_winding', 'ambient', 'stator_tooth', 'stator_yoke',
      'pm', 'torque'],
      dtype='object')

```

```

[11]: ## Visualización de resultados
def eva(x,y,coef):
    #plt.scatter(x,y)

```

```
yp=[]
temp=0
for a in range(len(x)):
    for b in range(len(coef)):
        temp+=coef[-1-b]*(x[a]**b)
    yp.append(temp)
    temp=0
plt.plot(x,yp)
return(yp)
```

```
[12]: # visualización de funcion
def vis(coef):
    p=str(coef[0])
    for a in range(len(coef)-1):
        p+='+'+str(coef[a+1])+'x^'+str(a+1)
    return p
```

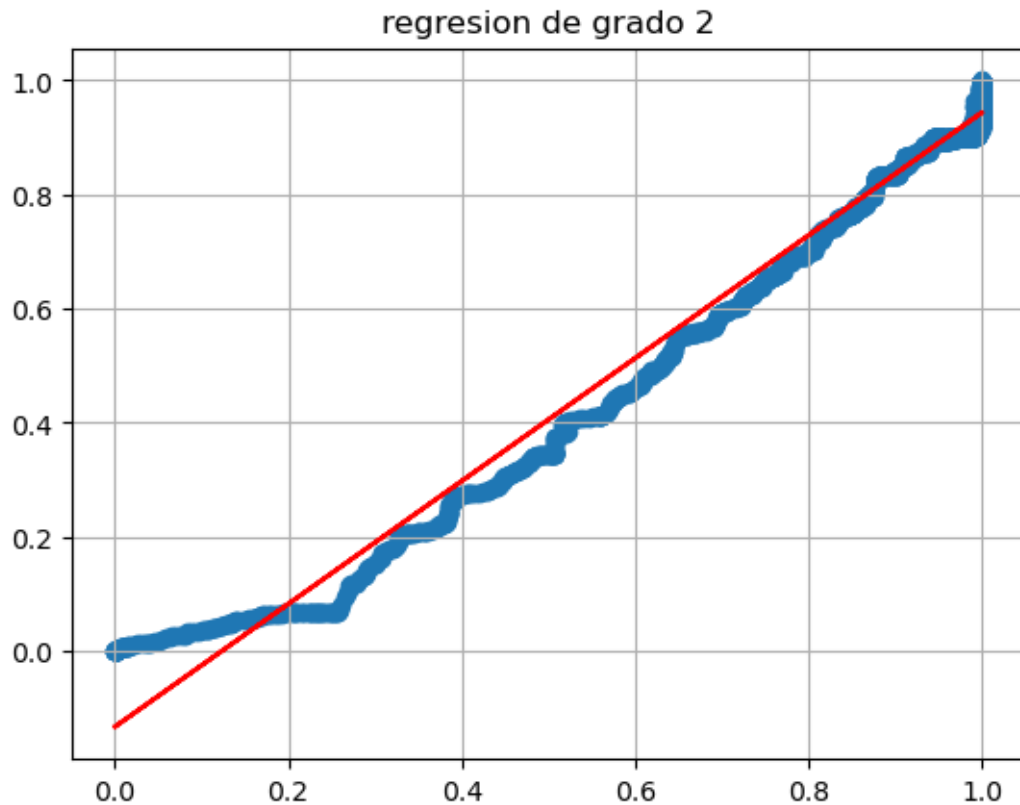
```
[13]: #np.cov(motor_db_norm['stator_winding'],motor_db_norm['stator_yoke'])
mm=polinomyal(motor_db_norm['stator_winding'],motor_db_norm['stator_yoke'],1)
plt.title('regresion de grado '+'2')
```

(3600,)

Resultado: $\begin{bmatrix} -0.13339356 \\ 1.07732748 \end{bmatrix}$

Coefficiente de determinación: 0.9913983235923303

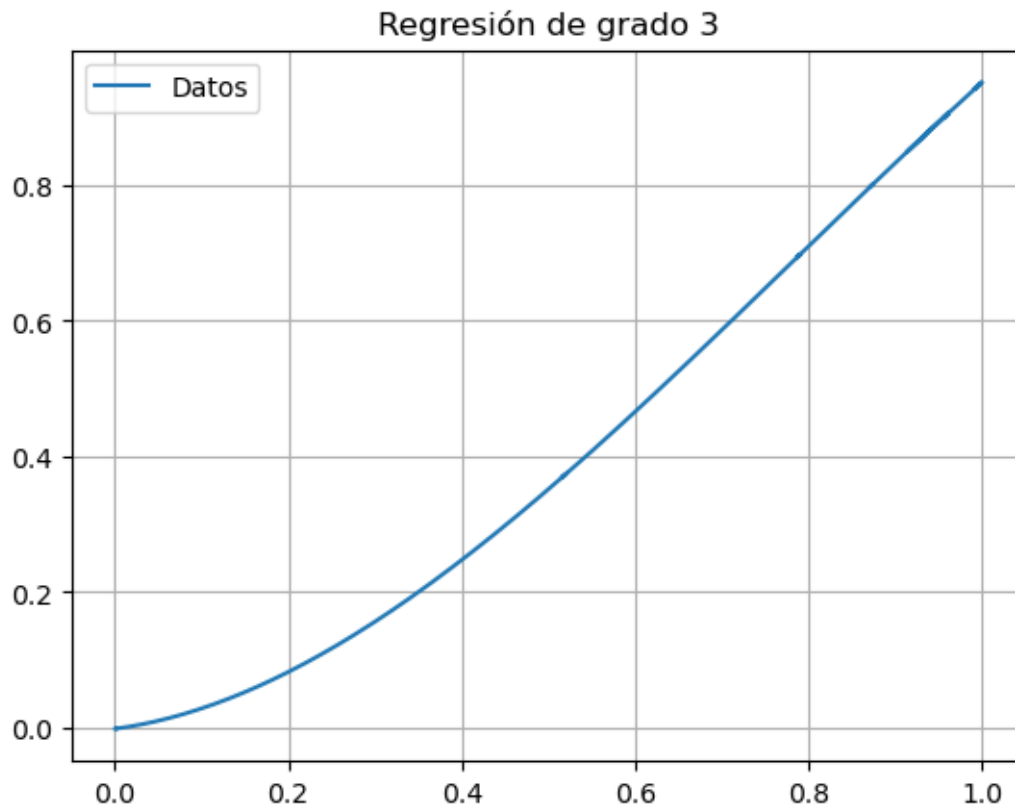
```
[13]: Text(0.5, 1.0, 'regresion de grado 2')
```



```
[14]: # evaluación de mínimos cuadrados graficas
```

```
[15]: grado=3
rres=np.
    ↪polyfit(motor_db_norm['stator_winding'],motor_db_norm['stator_yoke'],grado)
eva(motor_db_norm['stator_winding'],motor_db_norm['stator_yoke'],rres)
print(np.flip(rres))
print('f(x)='+vis(np.flip(rres)))
plt.title('Regresión de grado '+str(grado))
plt.legend(['Datos', 'Regresión'])
plt.grid()

[-0.00180375  0.16912855  1.36645506 -0.58210017]
f(x)=-0.001803747941988713+0.16912855425541518x^1+1.3664550558138868x^2+-0.58210
01650586674x^3
```

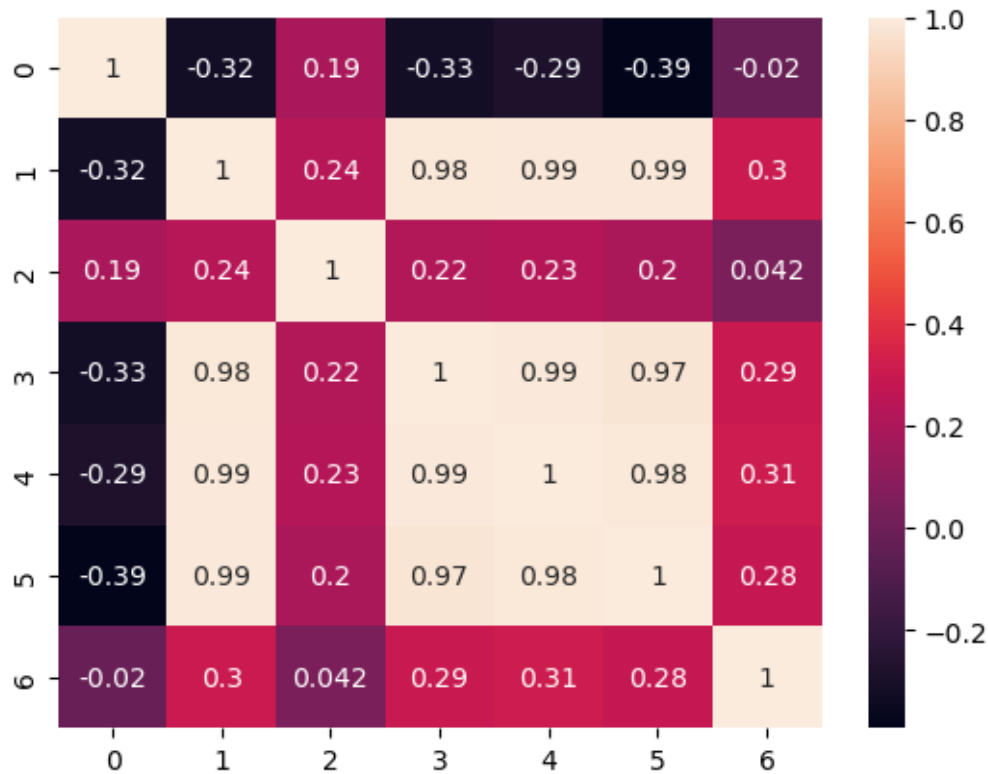


```
[16]: dim=len(motor_db_norm.columns)
      col_names=motor_db_norm.columns
      m_cc=np.zeros((dim,dim))

      for a in range(dim):
          for b in range(dim):
              ax=motor_db_norm[col_names[a]][inicio:fin].reset_index(drop=True)
              ay=motor_db_norm[col_names[b]][inicio:fin].reset_index(drop=True)
              m_cc[a,b]=cc(ax,ay)

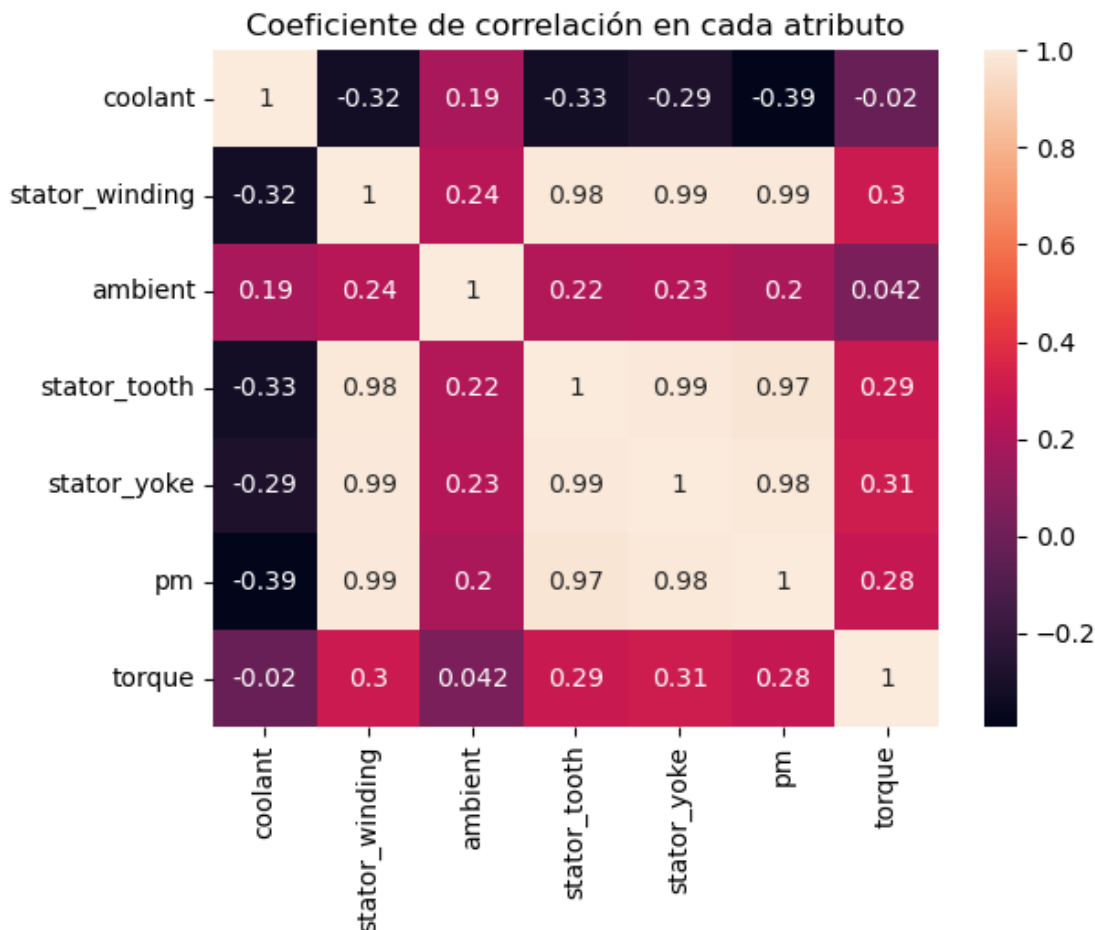
      sns.heatmap(m_cc,annot=True)
```

[16]: <AxesSubplot:>



```
[17]: sns.heatmap(motor_db[inicio:fin].corr(),annot=True)
plt.title('Coeficiente de correlación en cada atributo')
```

```
[17]: Text(0.5, 1.0, 'Coeficiente de correlación en cada atributo')
```



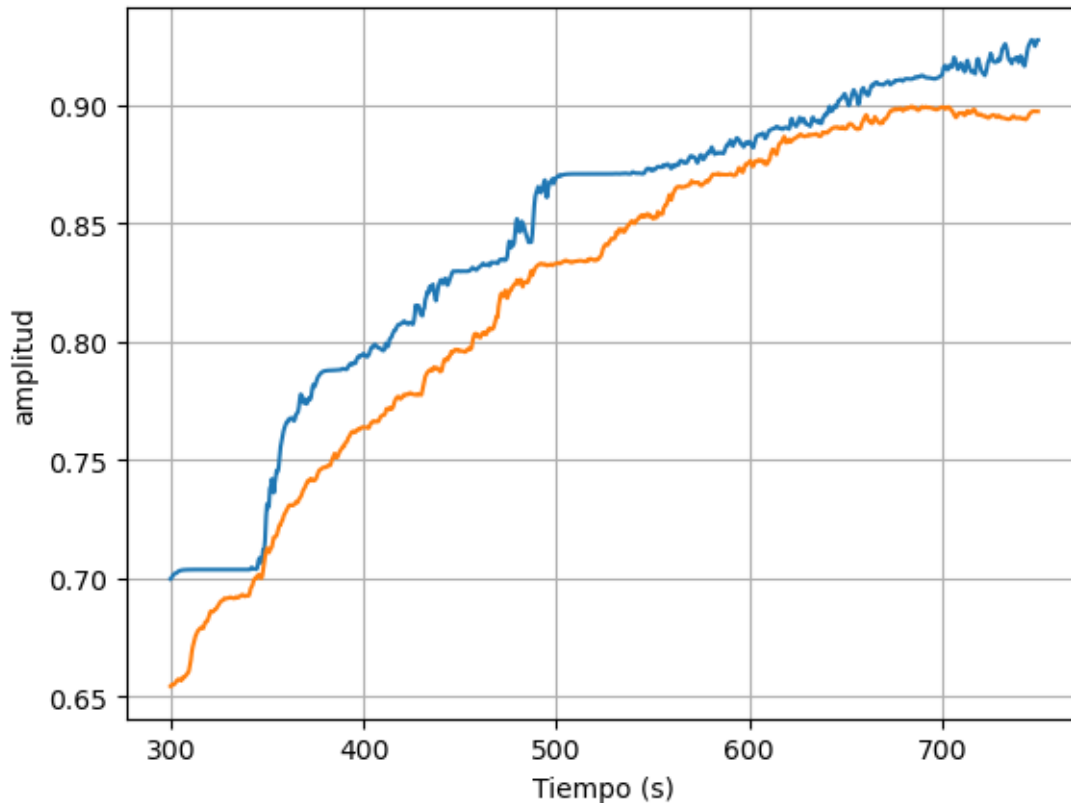
```
[18]: for a in range(dim):
        #ax=motor_db_norm[col_names[a]][inicio:fin].reset_index(drop=True)
        ay=motor_db_norm[col_names[a]][inicio:fin].reset_index(drop=True)
        print('coeficiente de correlación',motor_db_norm.columns[a],':\n',
              '\n',cc(tiempo,ay))
```

```
coeficiente de correlación coolant : -0.4585024723548856
coeficiente de correlación stator_winding : 0.9700502488743766
coeficiente de correlación ambient : 0.17304071962394926
coeficiente de correlación stator_tooth : 0.943709093712959
coeficiente de correlación stator_yoke : 0.957736252124007
coeficiente de correlación pm : 0.9925641786192424
coeficiente de correlación torque : 0.2611378518819429
```

```
[19]: print(np.flip(rres))
```

```
[-0.00180375  0.16912855  1.36645506 -0.58210017]
```

```
[20]: plt.plot(tiempo,motor_db_norm['stator_tooth'][inicio:fin])
plt.plot(tiempo,motor_db_norm['stator_yoke'][inicio:fin])
plt.xlabel('Tiempo (s)')
plt.ylabel('amplitud')
plt.grid()
```



4 Regresiones

```
[21]: grado=6
grados=[]
#plt.scatter(tiempo,tiempo,motor_db['coolant'][inicio:fin].
#reset_index(drop=True))
ii='coolant'
metrics=np.zeros((grado,6))
for tt in range(grado):
    r=np.polyfit(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),tt+1)
    pred=eva(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),r)
    grados.append('grado '+str(tt+1))
    print('grado '+str(tt+1))
    mse1=mse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
```

```

print('MSE:',mse1)
rmse1=rmse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('RMSE:',rmse1)
mae1=mae(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('MAE:',mae1)
rae1=rae(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
print('RAE:',rae1)
cc1=cc(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
print('CC:',cc1)
r21=r2(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('R2:',r21)
metrics[tt,:]=[mse1,rmse1,mae1,rae1,cc1,r21]
plt.grid()
plt.xlabel('Tiempo (s)')
plt.ylabel('Temperatura (°C)')
plt.title('Regresión de '+ ii + ' polinomial')
plt.legend(grados)
plt.scatter(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),s=1.2,c='k')

```

```

grado 1
MSE: 0.20075709804378178
RMSE: 0.4480592572905751
MAE: 0.3639740980154507
RAE: 0.9365264621794305
CC: 0.4585024723548854
R2: 0.21022451715554274
grado 2
MSE: 0.12109053471135209
RMSE: 0.34798065278309953
MAE: 0.2863977730206505
RAE: 0.7369180790763618
CC: 0.7236239425849015
R2: 0.5236316102821164
grado 3
MSE: 0.06151845289873675
RMSE: 0.24802913719709777
MAE: 0.20050880587043426
RAE: 0.5159207856315333
CC: 0.8706246650943814
R2: 0.7579873074707023
grado 4
MSE: 0.061452484522103855
RMSE: 0.24789611639173345
MAE: 0.2005687193830996
RAE: 0.5160749465741914
CC: 0.8707736939630276
R2: 0.7582468260980156
grado 5

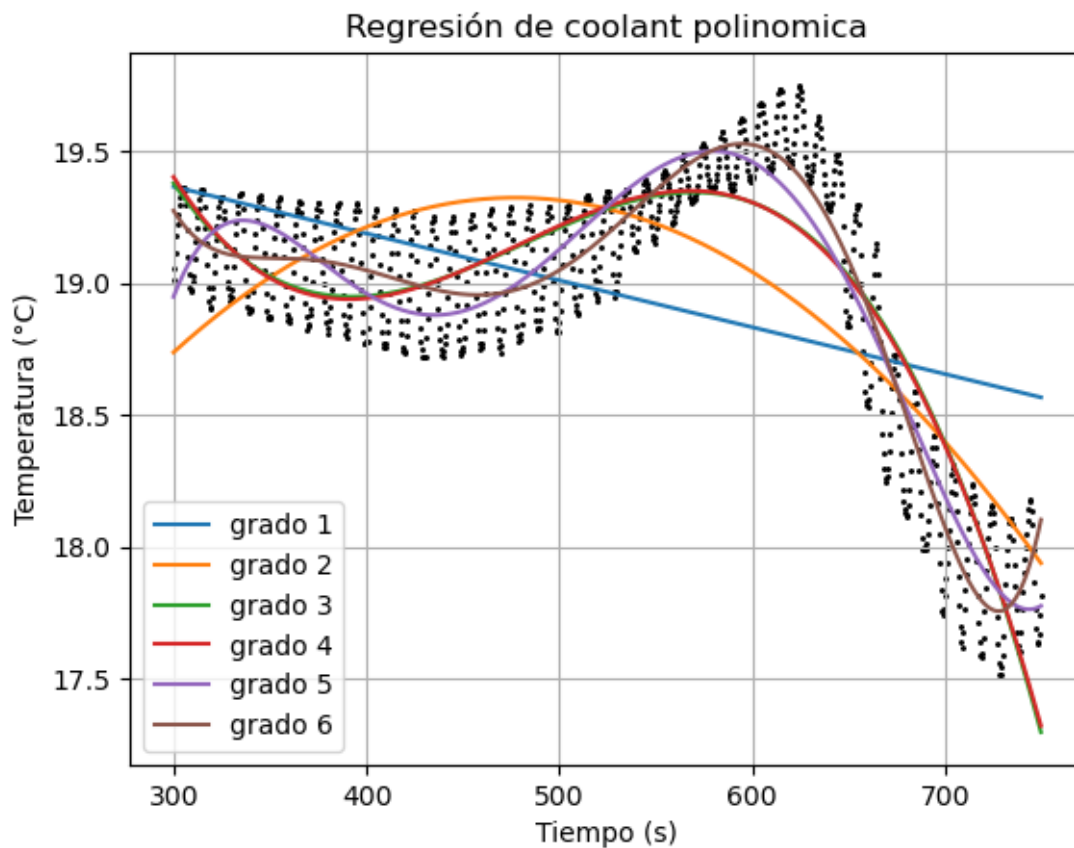
```

```

MSE: 0.042015116550607336
RMSE: 0.2049758926083927
MAE: 0.1718564911547315
RAE: 0.44219671823152384
CC: 0.9136263716437242
R2: 0.834713146962876
grado 6
MSE: 0.03340015613336865
RMSE: 0.18275709598636286
MAE: 0.15605522391600288
RAE: 0.4015391412618295
CC: 0.9319894178825563
R2: 0.8686042750450659

```

[21]: <matplotlib.collections.PathCollection at 0x21d4197fbb0>



```

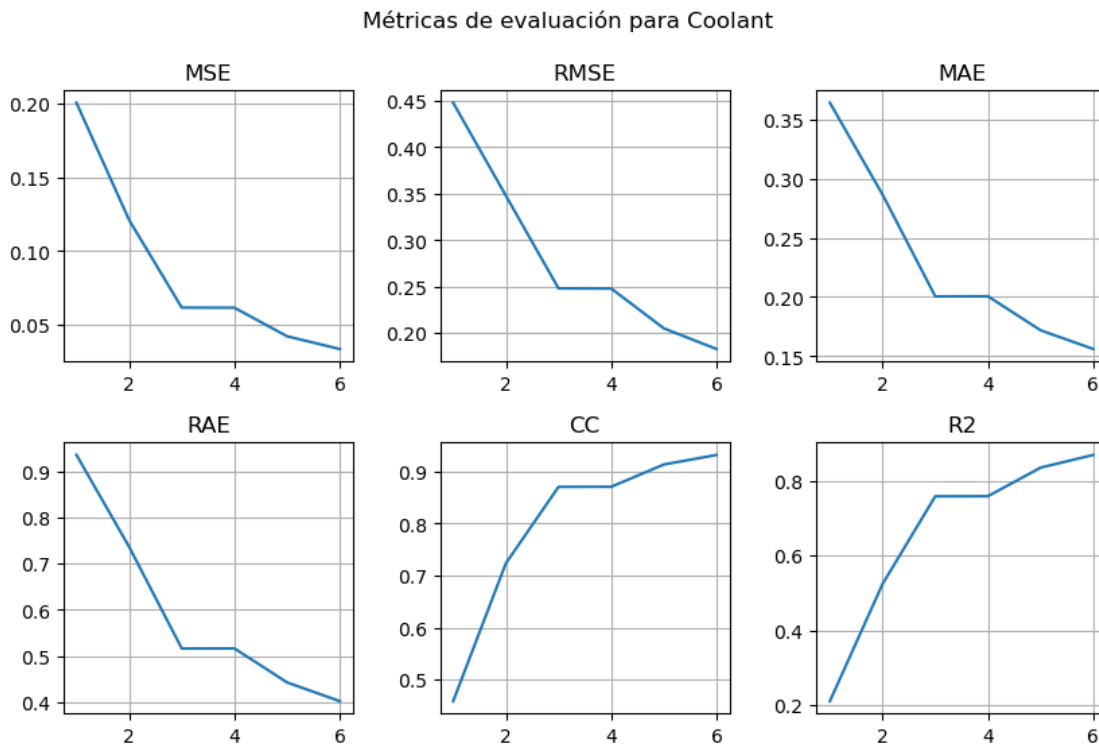
[22]: # Creamos una figura con 6 subplots
fig, axs = plt.subplots(2, 3, figsize=(10, 6))
gr=np.arange(1,grado+1,1)
# Dibujamos cada gráfica en su subplot correspondiente

```

```

axs[0, 0].plot(gr,metrics[:,0])
axs[0, 0].set_title('MSE')
axs[0, 0].grid(True)
axs[0, 1].plot(gr,metrics[:,1])
axs[0, 1].set_title('RMSE')
axs[0, 1].grid(True)
axs[0, 2].plot(gr,metrics[:,2])
axs[0, 2].set_title('MAE')
axs[0, 2].grid(True)
axs[1, 0].plot(gr,metrics[:,3])
axs[1, 0].set_title('RAE')
axs[1, 0].grid(True)
axs[1, 1].plot(gr,metrics[:,4])
axs[1, 1].set_title('CC')
axs[1, 1].grid(True)
axs[1, 2].plot(gr,metrics[:,5])
axs[1, 2].set_title('R2')
axs[1, 2].grid(True)
# Ajustamos los márgenes entre los subplots
plt.subplots_adjust(wspace=0.3, hspace=0.3)
fig.suptitle('Métricas de evaluación para Coolant')
# Mostramos la figura
plt.show()

```



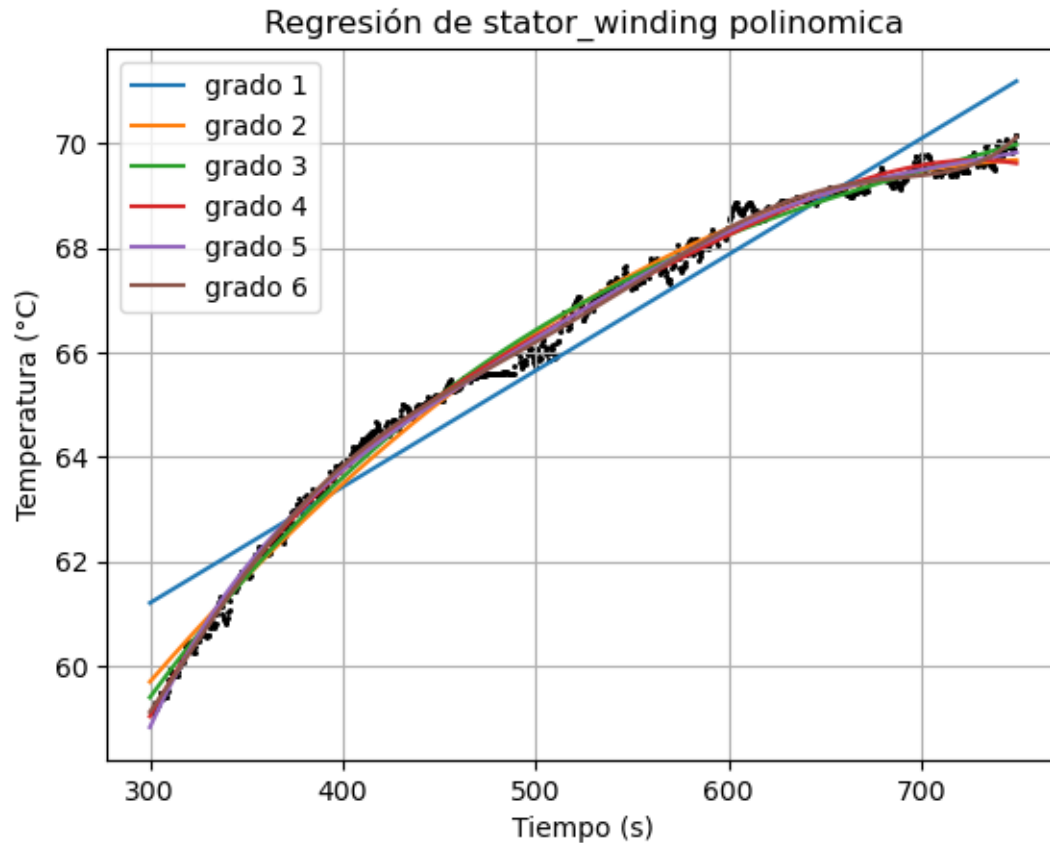
4.1 Temperatura de los devanados del estator

```
[23]: grado=6
      grados=[]
      #plt.scatter(tiempo,tiempo,motor_db['coolant'][inicio:fin].
      ↪reset_index(drop=True))
      ii='stator_winding' #'coolant', 'stator_winding', 'ambient', 'stator_tooth',
      ↪'stator_yoke', 'pm', 'torque'
      metrics=np.zeros((grado,6))
      for tt in range(grado):
          r=np.polyfit(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),tt+1)
          pred=eva(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),r)
          grados.append('grado '+str(tt+1))
          print('grado '+str(tt+1))
          mse1=mse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('MSE:',mse1)
          rmse1=rmse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('RMSE:',rmse1)
          mae1=mae(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('MAE:',mae1)
          rae1=rae(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
          print('RAE:',rae1)
          cc1=cc(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
          print('CC:',cc1)
          r21=r2(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('R2:',r21)
          metrics[tt,:]=[mse1,rmse1,mae1,rae1,cc1,r21]
      plt.grid()
      plt.xlabel('Tiempo (s)')
      plt.ylabel('Temperatura (°C)')
      plt.title('Regresión de '+ ii + ' polinomica')
      plt.legend(grados)
      plt.scatter(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),s=1.2,c='k')
```

```
grado 1
MSE: 0.5225236210548798
RMSE: 0.7228579535807016
MAE: 0.605606647247954
RAE: 0.2401524295646302
CC: 0.970050248874378
R2: 0.9409974853412413
grado 2
MSE: 0.06378782943459899
RMSE: 0.2525625257923253
MAE: 0.19902680579553977
RAE: 0.07892378853087
CC: 0.9963920826594234
R2: 0.9927971823863825
```

```
grado 3
MSE: 0.05086645102861406
RMSE: 0.22553591959733169
MAE: 0.1791617808393038
RAE: 0.07104634196009477
CC: 0.997123985947313
R2: 0.9942562433514568
grado 4
MSE: 0.0359464821833278
RMSE: 0.1895955753263451
MAE: 0.14748830277602426
RAE: 0.05848627059326863
CC: 0.9979684273000471
R2: 0.9959409818877263
grado 5
MSE: 0.03184489963462953
RMSE: 0.17845139291871479
MAE: 0.1409332468262722
RAE: 0.05588687275076029
CC: 0.9982004437468203
R2: 0.9964041258963457
grado 6
MSE: 0.024854387974187576
RMSE: 0.15765274489899495
MAE: 0.12288177586803975
RAE: 0.04872858836346894
CC: 0.9985957555819274
R2: 0.9971934830662373
```

[23]: <matplotlib.collections.PathCollection at 0x21d42b32f10>

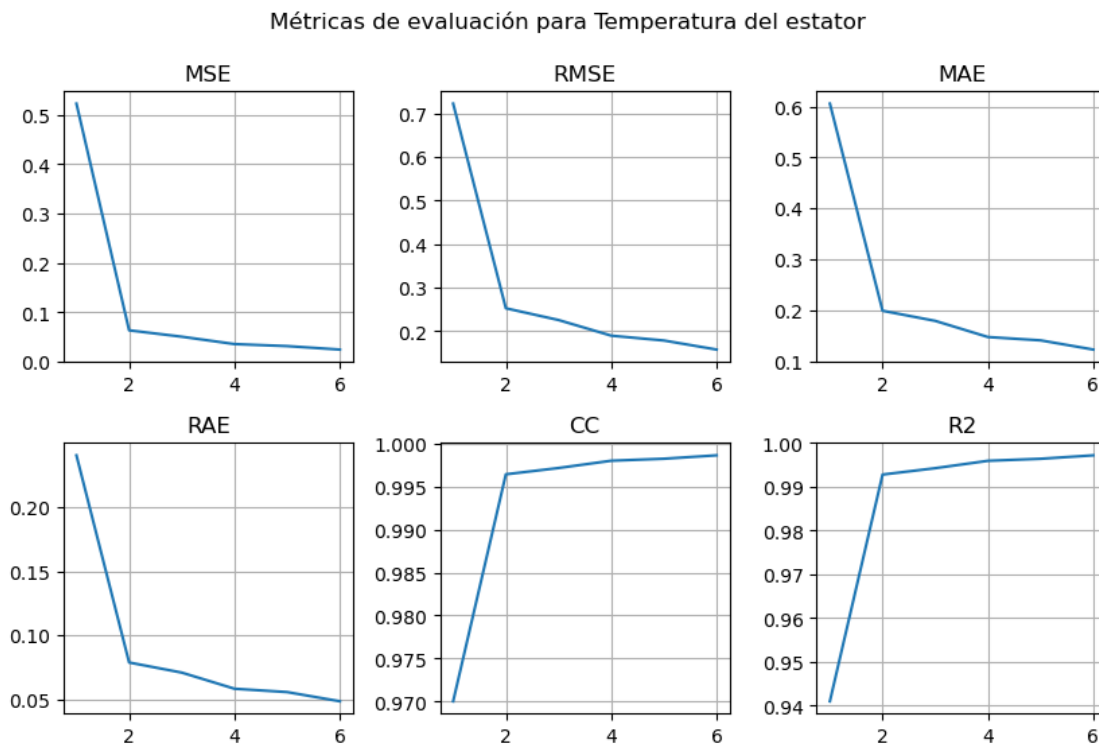


```
[24]: # Creamos una figura con 6 subplots
fig, axs = plt.subplots(2, 3, figsize=(10, 6))
gr=np.arange(1,grado+1,1)
# Dibujamos cada gráfica en su subplot correspondiente
axs[0, 0].plot(gr,metrics[:,0])
axs[0, 0].set_title('MSE')
axs[0, 0].grid(True)
axs[0, 1].plot(gr,metrics[:,1])
axs[0, 1].set_title('RMSE')
axs[0, 1].grid(True)
axs[0, 2].plot(gr,metrics[:,2])
axs[0, 2].set_title('MAE')
axs[0, 2].grid(True)
axs[1, 0].plot(gr,metrics[:,3])
axs[1, 0].set_title('RAE')
axs[1, 0].grid(True)
axs[1, 1].plot(gr,metrics[:,4])
axs[1, 1].set_title('CC')
axs[1, 1].grid(True)
axs[1, 2].plot(gr,metrics[:,5])
```

```

axs[1, 2].set_title('R2')
axs[1, 2].grid(True)
# Ajustamos los márgenes entre los subplots
plt.subplots_adjust(wspace=0.3, hspace=0.3)
fig.suptitle('Métricas de evaluación para Temperatura del estator')
# Mostramos la figura
plt.show()

```



4.2 Temperatura ambiente

```

[25]: grado=10
      grados=[]
      #plt.scatter(tiempo,tiempo,motor_db['coolant'][inicio:fin].
      ↪reset_index(drop=True))
      ii='ambient' #'coolant', 'stator_winding', 'ambient', 'stator_tooth',
      ↪'stator_yoke', 'pm', 'torque'
      metrics=np.zeros((grado,6))
      for tt in range(grado):
          r=np.polyfit(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),tt+1)
          pred=eva(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),r)
          grados.append('grado '+str(tt+1))
          print('grado '+str(tt+1))

```

```

mse1=mse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('MSE:',mse1)
rmse1=rmse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('RMSE:',rmse1)
mae1=mae(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('MAE:',mae1)
rae1=rae(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
print('RAE:',rae1)
cc1=cc(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
print('CC:',cc1)
r21=r2(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('R2:',r21)
metrics[tt,:]=[mse1,rmse1,mae1,rae1,cc1,r21]
plt.grid()
plt.xlabel('Tiempo (s)')
plt.ylabel('Temperatura (°C)')
plt.title('Regresión de '+ ii + ' polinomial')
plt.legend(grados)
plt.scatter(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),s=1.2,c='k')

```

grado 1

MSE: 0.15178443530802657
 RMSE: 0.38959521982183837
 MAE: 0.32226127092101237
 RAE: 0.9801665655724762
 CC: 0.17304071962394893
 R2: 0.02994309064797337

grado 2

MSE: 0.14387938984989132
 RMSE: 0.3793143681036764
 MAE: 0.31611003547353217
 RAE: 0.9614574128860456
 CC: 0.28366242386154217
 R2: 0.08046437071100529

grado 3

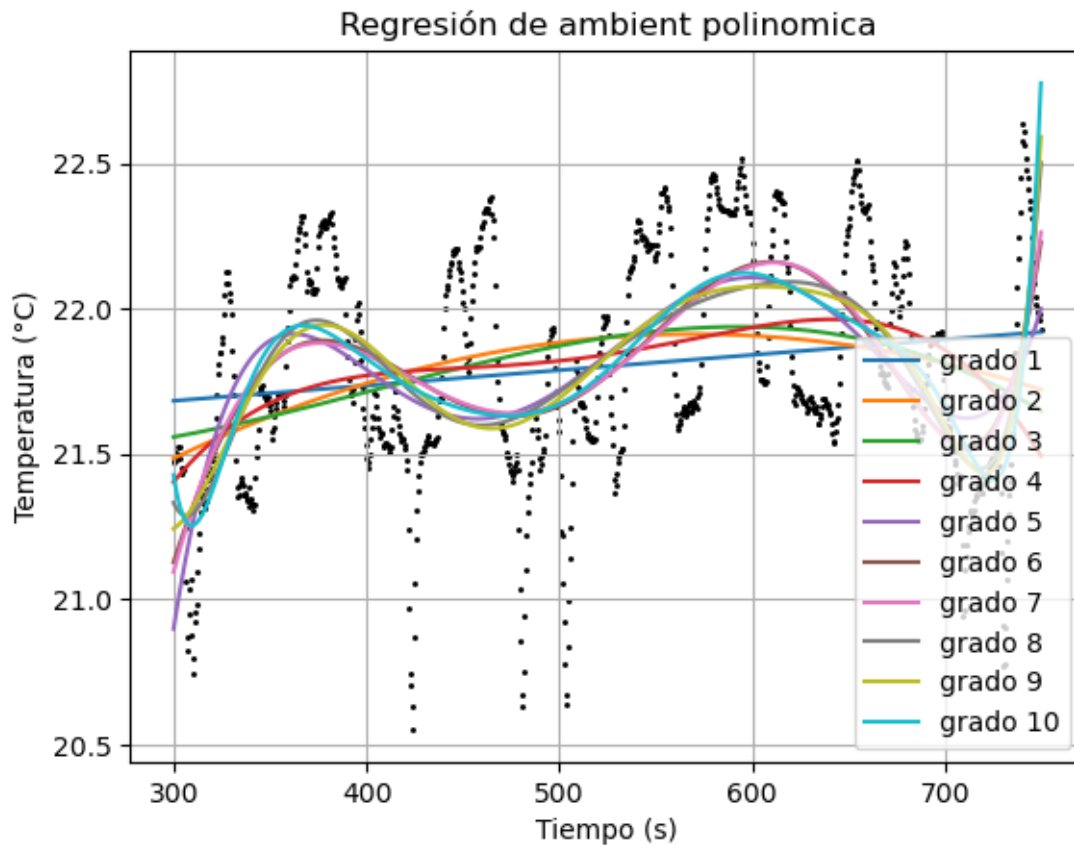
MSE: 0.14309324208174806
 RMSE: 0.37827667398578524
 MAE: 0.31597442918911156
 RAE: 0.9610449626226594
 CC: 0.2923844292657682
 R2: 0.08548865447706763

grado 4

MSE: 0.14038517905450468
 RMSE: 0.3746801022932826
 MAE: 0.3124300143755866
 RAE: 0.95026452696929
 CC: 0.3206180459271105
 R2: 0.10279593137411645

```
grado 5
MSE: 0.11631167069513303
RMSE: 0.3410449687286605
MAE: 0.2861880447018256
RAE: 0.8704488506533734
CC: 0.5066064873434308
R2: 0.25665013301844875
grado 6
MSE: 0.11207366739478837
RMSE: 0.3347740542437367
MAE: 0.28072531919637805
RAE: 0.853833819991957
CC: 0.532668080740474
R2: 0.28373528423973887
grado 7
MSE: 0.11199165462169357
RMSE: 0.3346515420877268
MAE: 0.28099943246029624
RAE: 0.8546675430630064
CC: 0.5331598533914644
R2: 0.28425942926840586
grado 8
MSE: 0.10834739415912577
RMSE: 0.329161653536869
MAE: 0.27173019161779033
RAE: 0.8264748907591868
CC: 0.5545718708231119
R2: 0.30754995990824524
grado 9
MSE: 0.1078820003299676
RMSE: 0.3284539546572207
MAE: 0.27119835724642316
RAE: 0.8248573018142089
CC: 0.5572470724832711
R2: 0.31052429979117524
grado 10
MSE: 0.10604550411675957
RMSE: 0.32564628681555635
MAE: 0.26984341871195044
RAE: 0.820736218799485
CC: 0.5676807010512821
R2: 0.32226137834607416
```

[25]: <matplotlib.collections.PathCollection at 0x21d42effb80>

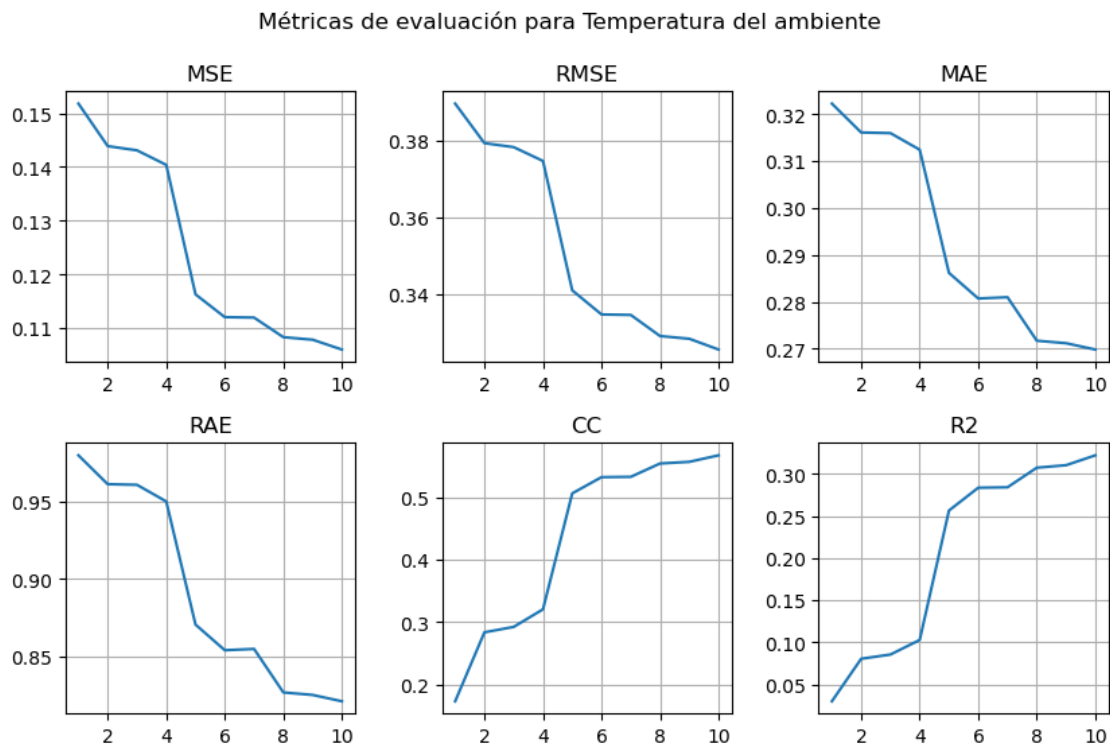


```
[26]: # Creamos una figura con 6 subplots
fig, axs = plt.subplots(2, 3, figsize=(10, 6))
gr=np.arange(1,grado+1,1)
# Dibujamos cada gráfica en su subplot correspondiente
axs[0, 0].plot(gr,metrics[:,0])
axs[0, 0].set_title('MSE')
axs[0, 0].grid(True)
axs[0, 1].plot(gr,metrics[:,1])
axs[0, 1].set_title('RMSE')
axs[0, 1].grid(True)
axs[0, 2].plot(gr,metrics[:,2])
axs[0, 2].set_title('MAE')
axs[0, 2].grid(True)
axs[1, 0].plot(gr,metrics[:,3])
axs[1, 0].set_title('RAE')
axs[1, 0].grid(True)
axs[1, 1].plot(gr,metrics[:,4])
axs[1, 1].set_title('CC')
axs[1, 1].grid(True)
axs[1, 2].plot(gr,metrics[:,5])
```

```

axs[1, 2].set_title('R2')
axs[1, 2].grid(True)
# Ajustamos los márgenes entre los subplots
plt.subplots_adjust(wspace=0.3, hspace=0.3)
fig.suptitle('Métricas de evaluación para Temperatura del ambiente')
# Mostramos la figura
plt.show()

```



4.3 Temperatura de los dientes del estator

```

[35]: grado=4
      grados=[]
      #plt.scatter(tiempo,tiempo,motor_db['coolant'][inicio:fin].
      ↪reset_index(drop=True))
      ii='stator_tooth' # 'coolant', 'stator_winding', 'ambient', 'stator_tooth',
      ↪ 'stator_yoke', 'pm', 'torque'
      metrics=np.zeros((grado,6))
      for tt in range(grado):
          r=np.polyfit(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),tt+1)
          pred=eva(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),r)
          grados.append('grado '+str(tt+1))
          print('grado '+str(tt+1))

```

```

mse1=mse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('MSE:',mse1)
rmse1=rmse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('RMSE:',rmse1)
mae1=mae(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('MAE:',mae1)
rae1=rae(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
print('RAE:',rae1)
cc1=cc(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
print('CC:',cc1)
r21=r2(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('R2:',r21)
metrics[tt,:]=[mse1,rmse1,mae1,rae1,cc1,r21]
plt.grid()
plt.xlabel('Tiempo (s)')
plt.ylabel('Temperatura (°C)')
plt.title('Regresión de '+ ii + ' polinomial')
plt.legend(grados)
plt.scatter(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),s=1.2,c='k')

```

grado 1

MSE: 0.7555335502142282
 RMSE: 0.8692143292734124
 MAE: 0.7039535691668678
 RAE: 0.32106988824399807
 CC: 0.9437090937129581
 R2: 0.8905868535565329

grado 2

MSE: 0.16380826512161503
 RMSE: 0.40473233762774014
 MAE: 0.3206737771741262
 RAE: 0.1462577907260695
 CC: 0.9880678041895997
 R2: 0.9762779856760568

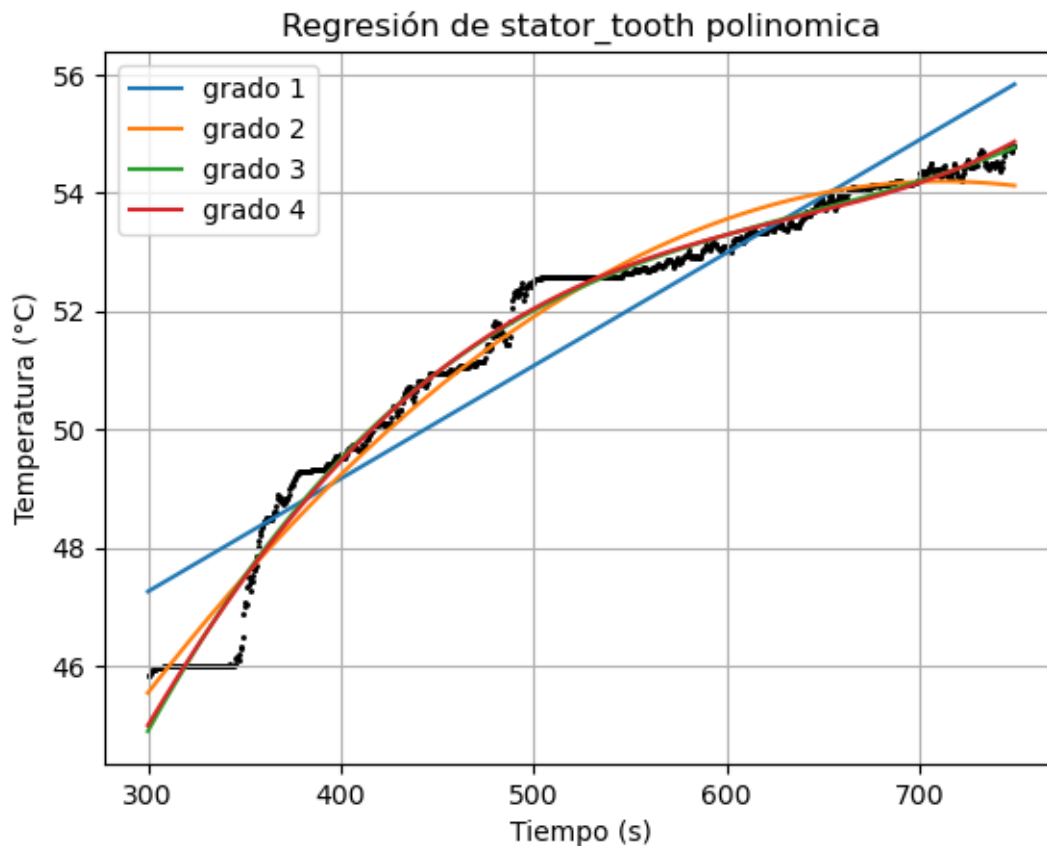
grado 3

MSE: 0.10333380558827142
 RMSE: 0.32145575992393016
 MAE: 0.22342981355339975
 RAE: 0.10190527956675886
 CC: 0.9924896165634534
 R2: 0.9850356389862707

grado 4

MSE: 0.10226818239648078
 RMSE: 0.31979396866807974
 MAE: 0.22905117493680296
 RAE: 0.10446915586514094
 CC: 0.9925673569043544
 R2: 0.9851899579920962

[35]: <matplotlib.collections.PathCollection at 0x21d433a4a00>



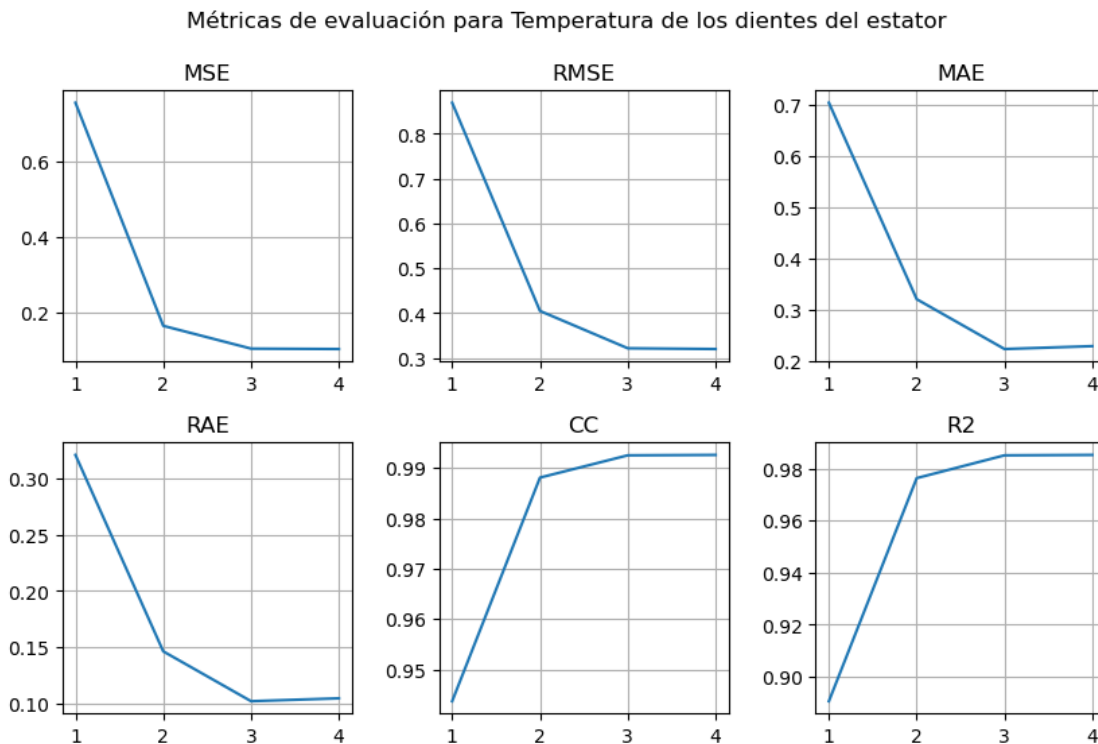
```
[36]: # Creamos una figura con 6 subplots
fig, axs = plt.subplots(2, 3, figsize=(10, 6))
gr=np.arange(1,grado+1,1)
# Dibujamos cada gráfica en su subplot correspondiente
axs[0, 0].plot(gr,metrics[:,0])
axs[0, 0].set_title('MSE')
axs[0, 0].grid(True)
axs[0, 1].plot(gr,metrics[:,1])
axs[0, 1].set_title('RMSE')
axs[0, 1].grid(True)
axs[0, 2].plot(gr,metrics[:,2])
axs[0, 2].set_title('MAE')
axs[0, 2].grid(True)
axs[1, 0].plot(gr,metrics[:,3])
axs[1, 0].set_title('RAE')
axs[1, 0].grid(True)
axs[1, 1].plot(gr,metrics[:,4])
axs[1, 1].set_title('CC')
```



```

axs[1, 1].grid(True)
axs[1, 2].plot(gr,metrics[:,5])
axs[1, 2].set_title('R2')
axs[1, 2].grid(True)
# Ajustamos los márgenes entre los subplots
plt.subplots_adjust(wspace=0.3, hspace=0.3)
fig.suptitle('Métricas de evaluación para Temperatura de los dientes del_
↪estator')
# Mostramos la figura
plt.show()

```



4.4 Temperatura del yugo del estator

```

[29]: grado=5
      grados=[]
      #plt.scatter(tiempo,tiempo,motor_db['coolant'][inicio:fin].
      ↪reset_index(drop=True))
      ii='stator_yoke' #'coolant', 'stator_winding', 'ambient', 'stator_tooth',
      ↪'stator_yoke', 'pm', 'torque'
      metrics=np.zeros((grado,6))
      for tt in range(grado):
          r=np.polyfit(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),tt+1)

```

```

pred=eva(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),r)
grados.append('grado '+str(tt+1))
print('grado '+str(tt+1))
mse1=mse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('MSE:',mse1)
rmse1=rmse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('RMSE:',rmse1)
mae1=mae(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('MAE:',mae1)
rae1=rae(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
print('RAE:',rae1)
cc1=cc(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
print('CC:',cc1)
r21=r2(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('R2:',r21)
metrics[tt,:]=[mse1,rmse1,mae1,rae1,cc1,r21]
plt.grid()
plt.xlabel('Tiempo (s)')
plt.ylabel('Temperatura (°C)')
plt.title('Regresión de '+ ii + ' polinomial')
plt.legend(grados)
plt.scatter(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),s=1.2,c='k')

```

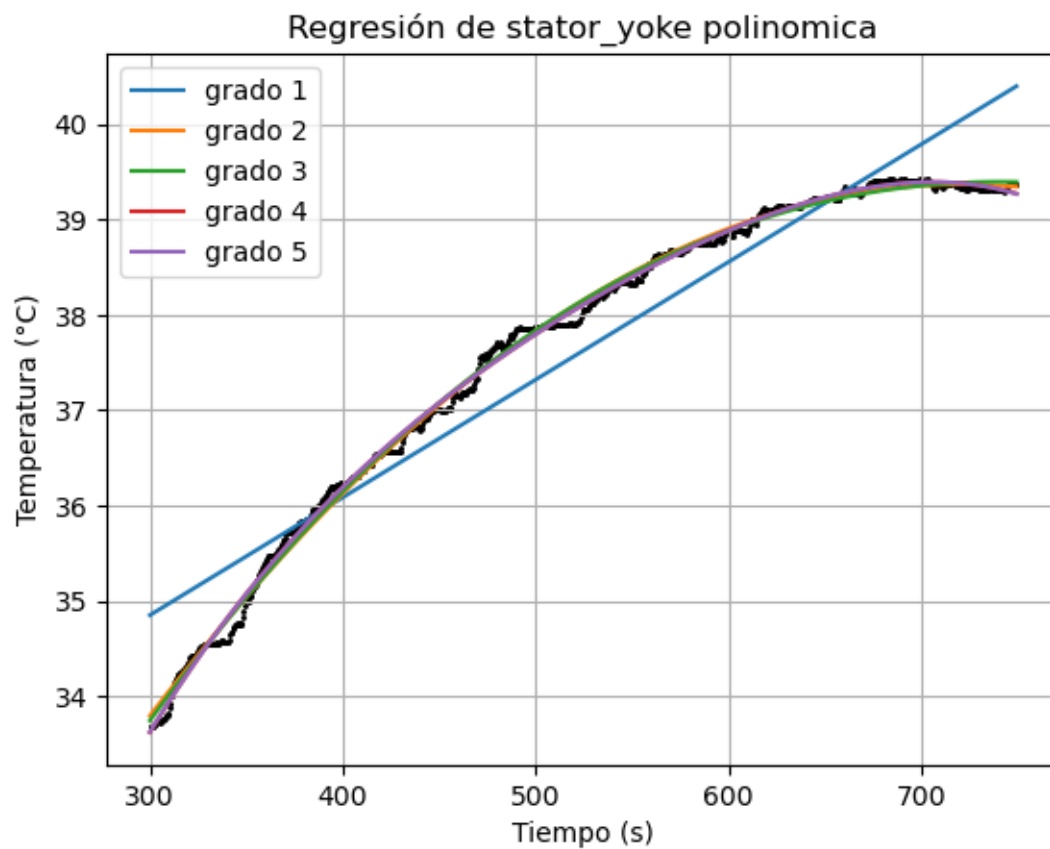
```

grado 1
MSE: 0.231995310114849
RMSE: 0.48165891470505245
MAE: 0.4002658922284293
RAE: 0.28049588331948583
CC: 0.9577362521240061
R2: 0.9172587286325399
grado 2
MSE: 0.008049787808809862
RMSE: 0.08972060972156766
MAE: 0.07206528867040331
RAE: 0.050501472133285076
CC: 0.998563487306113
R2: 0.9971290381809464
grado 3
MSE: 0.007677869640159826
RMSE: 0.08762345371052105
MAE: 0.07305191741261628
RAE: 0.05119287578752771
CC: 0.9986299029303465
R2: 0.9972616830266701
grado 4
MSE: 0.005956246333855493
RMSE: 0.07717672145054812
MAE: 0.06004829816326324

```

RAE: 0.042080279039923826
 CC: 0.9989372858431061
 R2: 0.9978757010475902
 grado 5
 MSE: 0.0059547529069679875
 RMSE: 0.0771670454725849
 MAE: 0.06010191982514009
 RAE: 0.04211785570010206
 CC: 0.998937552442202
 R2: 0.9978762336792165

[29]: <matplotlib.collections.PathCollection at 0x21d42dd1ee0>

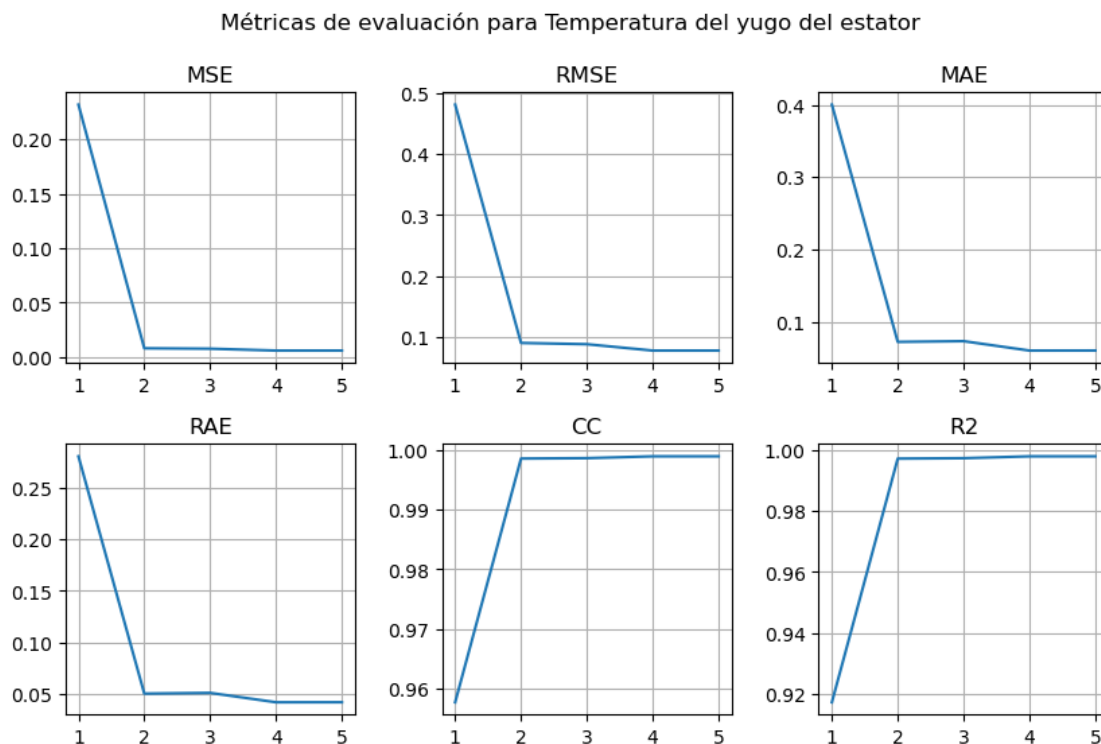


```
[30]: # Creamos una figura con 6 subplots
fig, axs = plt.subplots(2, 3, figsize=(10, 6))
gr=np.arange(1,grado+1,1)
# Dibujamos cada gráfica en su subplot correspondiente
axs[0, 0].plot(gr,metrics[:,0])
axs[0, 0].set_title('MSE')
axs[0, 0].grid(True)
```

```

axs[0, 1].plot(gr,metrics[:,1])
axs[0, 1].set_title('RMSE')
axs[0, 1].grid(True)
axs[0, 2].plot(gr,metrics[:,2])
axs[0, 2].set_title('MAE')
axs[0, 2].grid(True)
axs[1, 0].plot(gr,metrics[:,3])
axs[1, 0].set_title('RAE')
axs[1, 0].grid(True)
axs[1, 1].plot(gr,metrics[:,4])
axs[1, 1].set_title('CC')
axs[1, 1].grid(True)
axs[1, 2].plot(gr,metrics[:,5])
axs[1, 2].set_title('R2')
axs[1, 2].grid(True)
# Ajustamos los márgenes entre los subplots
plt.subplots_adjust(wspace=0.3, hspace=0.3)
fig.suptitle('Métricas de evaluación para Temperatura del yugo del estator')
# Mostramos la figura
plt.show()

```



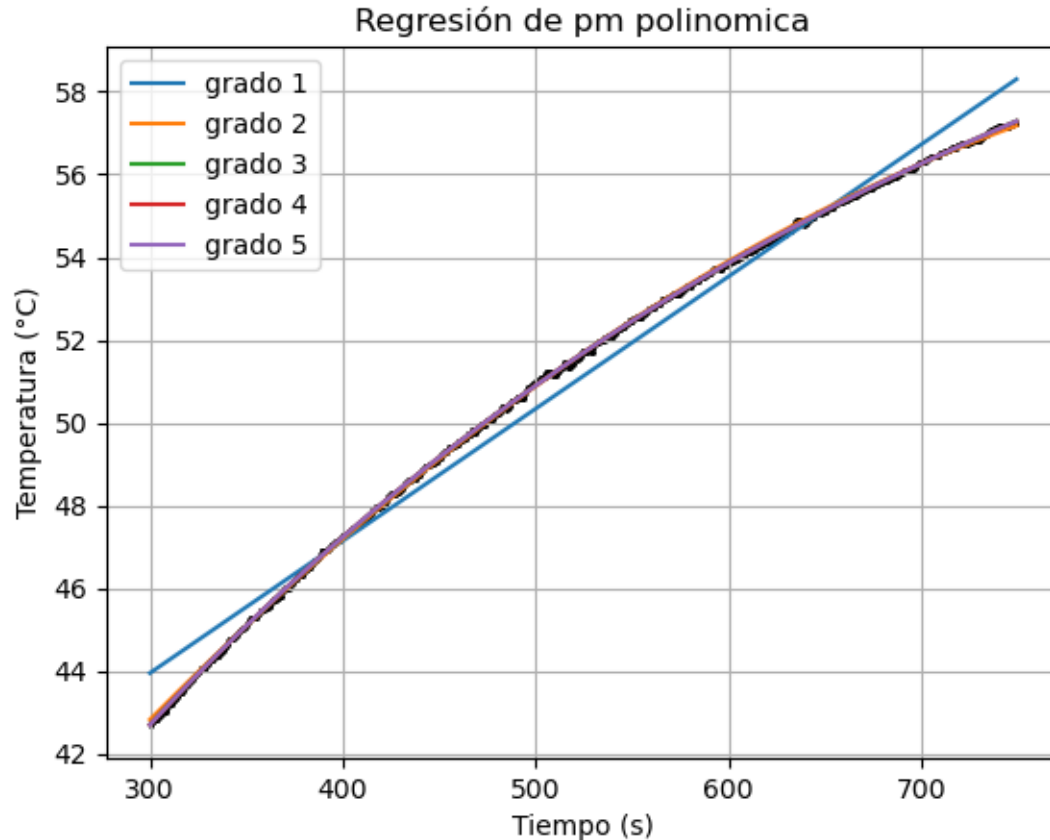
4.5 Temperatura del imán permanente

```
[31]: grado=5
      grados=[]
      #plt.scatter(tiempo,tiempo,motor_db['coolant'][inicio:fin].
      ↪reset_index(drop=True))
      ii='pm' #'coolant', 'stator_winding', 'ambient', 'stator_tooth',
      ↪'stator_yoke','pm', 'torque'
      metrics=np.zeros((grado,6))
      for tt in range(grado):
          r=np.polyfit(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),tt+1)
          pred=eva(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),r)
          grados.append('grado '+str(tt+1))
          print('grado '+str(tt+1))
          mse1=mse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('MSE:',mse1)
          rmse1=rmse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('RMSE:',rmse1)
          mae1=mae(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('MAE:',mae1)
          rae1=rae(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
          print('RAE:',rae1)
          cc1=cc(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
          print('CC:',cc1)
          r21=r2(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('R2:',r21)
          metrics[tt,:]=[mse1,rmse1,mae1,rae1,cc1,r21]
      plt.grid()
      plt.xlabel('Tiempo (s)')
      plt.ylabel('Temperatura (°C)')
      plt.title('Regresión de '+ ii + ' polinomial')
      plt.legend(grados)
      plt.scatter(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),s=1.2,c='k')
```

```
grado 1
MSE: 0.2587673815492282
RMSE: 0.5086918335782757
MAE: 0.4351372309581383
RAE: 0.12091978854632464
CC: 0.9925641786192438
R2: 0.9851836486780939
grado 2
MSE: 0.003632514241512242
RMSE: 0.0602703429682646
MAE: 0.048928210844097327
RAE: 0.013596604676163448
CC: 0.9998960004045554
R2: 0.9997920116250284
```

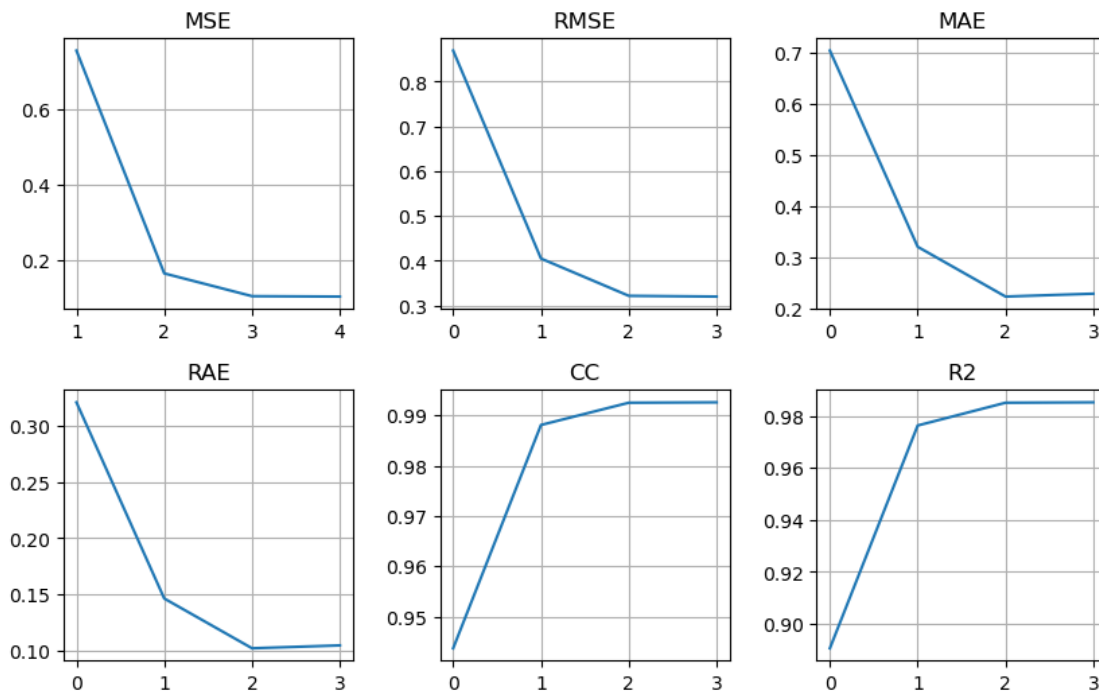
```
grado 3
MSE: 0.0017180715796593457
RMSE: 0.04144962701471927
MAE: 0.032453392607420986
RAE: 0.0090184362369069
CC: 0.9999508126216886
R2: 0.9999016276627757
grado 4
MSE: 0.001710749695163902
RMSE: 0.04136121003021916
MAE: 0.03234770934575522
RAE: 0.008989068035924943
CC: 0.9999510222480691
R2: 0.9999020468949539
grado 5
MSE: 0.001709596291825432
RMSE: 0.041347264623254484
MAE: 0.03233185382695085
RAE: 0.008984661963898206
CC: 0.9999510552701312
R2: 0.9999021129358474
```

[31]: <matplotlib.collections.PathCollection at 0x21d434bec40>



```
[37]: # Creamos una figura con 6 subplots
fig, axs = plt.subplots(2, 3, figsize=(10, 6))
gr=np.arange(1,grado+1,1)
# Dibujamos cada gráfica en su subplot correspondiente
axs[0, 0].plot(gr,metrics[:,0])
axs[0, 0].set_title('MSE')
axs[0, 0].grid(True)
axs[0, 1].plot(metrics[:,1])
axs[0, 1].set_title('RMSE')
axs[0, 1].grid(True)
axs[0, 2].plot(metrics[:,2])
axs[0, 2].set_title('MAE')
axs[0, 2].grid(True)
axs[1, 0].plot(metrics[:,3])
axs[1, 0].set_title('RAE')
axs[1, 0].grid(True)
axs[1, 1].plot(metrics[:,4])
axs[1, 1].set_title('CC')
axs[1, 1].grid(True)
axs[1, 2].plot(metrics[:,5])
axs[1, 2].set_title('R2')
axs[1, 2].grid(True)
# Ajustamos los márgenes entre los subplots
plt.subplots_adjust(wspace=0.3, hspace=0.3)
fig.suptitle('Métricas de evaluación para Temperatura del imán permanente')
# Mostramos la figura
plt.show()
```

Métricas de evaluación para Temperatura del imán permanente



4.6 Par del motor

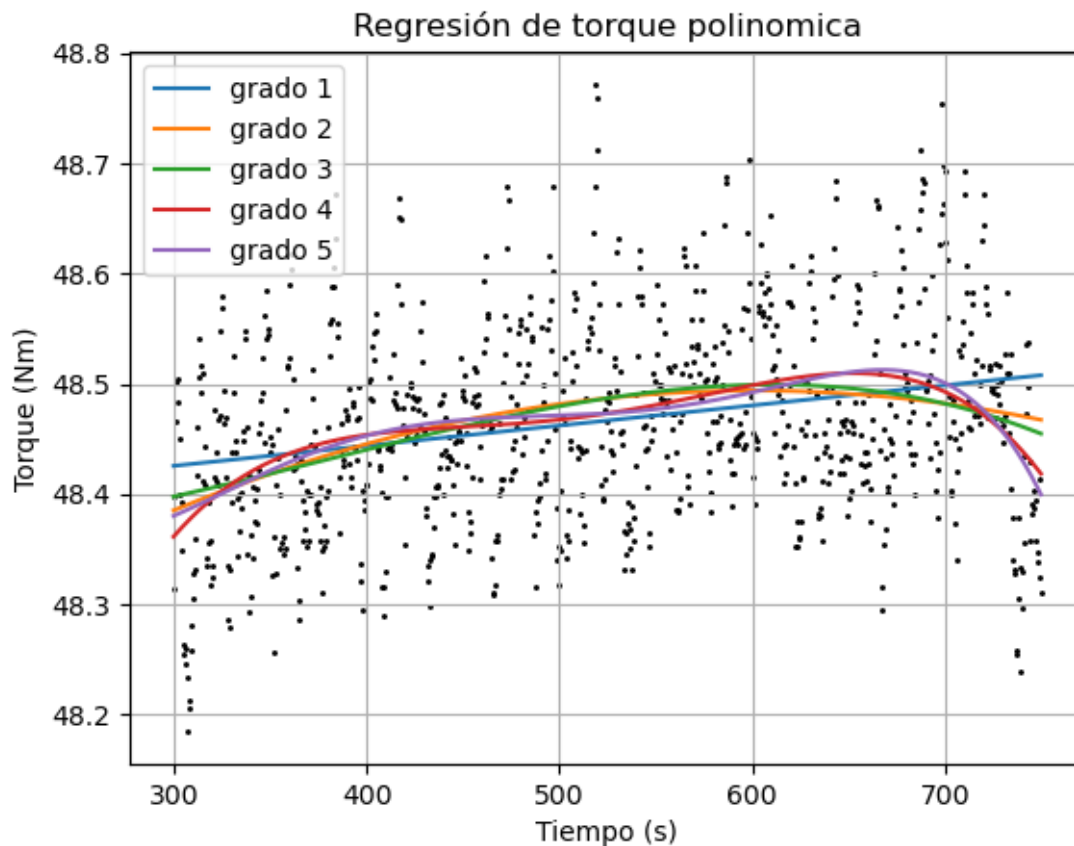
```
[38]: grado=5
      grados=[]
      #plt.scatter(tiempo,tiempo,motor_db['coolant'][inicio:fin].
      ↪reset_index(drop=True))
      ii='torque' #'coolant', 'stator_winding', 'ambient', 'stator_tooth',
      ↪'stator_yoke','pm', 'torque'
      metrics=np.zeros((grado,6))
      for tt in range(grado):
          r=np.polyfit(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),tt+1)
          pred=eva(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),r)
          grados.append('grado '+str(tt+1))
          print('grado '+str(tt+1))
          mse1=mse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('MSE:',mse1)
          rmse1=rmse(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('RMSE:',rmse1)
          mae1=mae(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
          print('MAE:',mae1)
          rae1=rae(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
          print('RAE:',rae1)
          cc1=cc(motor_db[ii][inicio:fin].reset_index(drop=True),np.array(pred))
```



```
print('CC:',cc1)
r21=r2(motor_db[ii][inicio:fin].reset_index(drop=True),pred)
print('R2:',r21)
metrics[tt,:]=[mse1,rmse1,mae1,rae1,cc1,r21]
plt.grid()
plt.xlabel('Tiempo (s)')
plt.ylabel('Torque (Nm)')
plt.title('Regresión de '+ ii + ' polinomial')
plt.legend(grados)
plt.scatter(tiempo,motor_db[ii][inicio:fin].reset_index(drop=True),s=1.2,c='k')
```

```
grado 1
MSE: 0.007712002443650902
RMSE: 0.08781800751355556
MAE: 0.06918721313398322
RAE: 0.973668097109817
CC: 0.2611378518819438
R2: 0.06819297768551552
grado 2
MSE: 0.007383140302425495
RMSE: 0.08592520178868068
MAE: 0.06819658661676932
RAE: 0.9597270610097748
CC: 0.3285238658263261
R2: 0.10792793041747344
grado 3
MSE: 0.007360329533893981
RMSE: 0.08579236291124041
MAE: 0.06815685629210455
RAE: 0.959167938191242
CC: 0.3326921314428548
R2: 0.11068405432398938
grado 4
MSE: 0.007209997466952213
RMSE: 0.08491170394564117
MAE: 0.06824057582181593
RAE: 0.9603461188331932
CC: 0.35895405090448584
R2: 0.12884801066074006
grado 5
MSE: 0.007176799772629185
RMSE: 0.08471599478628097
MAE: 0.06808875530882277
RAE: 0.9582095565510554
CC: 0.3644984783286814
R2: 0.13285914070392552
```

[38]: <matplotlib.collections.PathCollection at 0x21d43389850>



```
[34]: # Creamos una figura con 6 subplots
fig, axs = plt.subplots(2, 3, figsize=(10, 6))
gr=np.arange(1,grado+1,1)
# Dibujamos cada gráfica en su subplot correspondiente
axs[0, 0].plot(gr,metrics[:,0])
axs[0, 0].set_title('MSE')
axs[0, 0].grid(True)
axs[0, 1].plot(metrics[:,1])
axs[0, 1].set_title('RMSE')
axs[0, 1].grid(True)
axs[0, 2].plot(metrics[:,2])
axs[0, 2].set_title('MAE')
axs[0, 2].grid(True)
axs[1, 0].plot(metrics[:,3])
axs[1, 0].set_title('RAE')
axs[1, 0].grid(True)
axs[1, 1].plot(metrics[:,4])
axs[1, 1].set_title('CC')
```

```

axs[1, 1].grid(True)
axs[1, 2].plot(metrics[:,5])
axs[1, 2].set_title('R2')
axs[1, 2].grid(True)
# Ajustamos los márgenes entre los subplots
plt.subplots_adjust(wspace=0.3, hspace=0.3)
fig.suptitle('Métricas de evaluación para el Torque del motor')
# Mostramos la figura
plt.show()

```

