PECIER

Comité Nacional Peruano de la CIER

Curso Virtual

Python para el Análisis de Datos y la Automatización en el Sector Eléctrico

27, 29 y 31 de Octubre, 03, 05, 07, 10 y 12 de Noviembre 2025.

MARVIN COTO – FACILITADOR
E-mail: mcotoj@gmail.com

PECIER

Comité Nacional Peruano de la CIER

SESIÓN 4
Parte 1

PECIER
Comité Nacional Peruano de la CIER

Recordatorio de la sesión anterior

En la Sesión 3 se presentó una de los temas más importantes en programación:

Las funciones

Palabra clave: def

Hay varias formas de utilizarlas (con parámetros posicionales, nombras, o sin parámetros).

PECIER
Comité Nacional Peruano de la CIER

Repasemos:

```
1  def calculate(a):
2      square=a**2
3      print("Square is : ",square)
4
5  n=int(input("Enter Number "))
6  calculate(n)
```

Parameter(s)

Argument(s)

https://docs.google.com/forms/d/e/1FAIpQLSfIQhpjp8sleHFJBsovYEloI8fz5zinAlF8UwZvR212CsasTw/viewform?usp=header

PECIER
Comité Nacional Peruano de la CIER

Introducción a NumPy

NumPy (Numerical Python) es la biblioteca fundamental para computación científica en Python. Proporciona:

- Objetos **array** multidimensionales de alto rendimiento.
- Funciones matemáticas para operar en estos arrays.
- Herramientas para álgebra lineal, transformadas de Fourier, etc.

¿Por qué NumPy para el sector eléctrico?

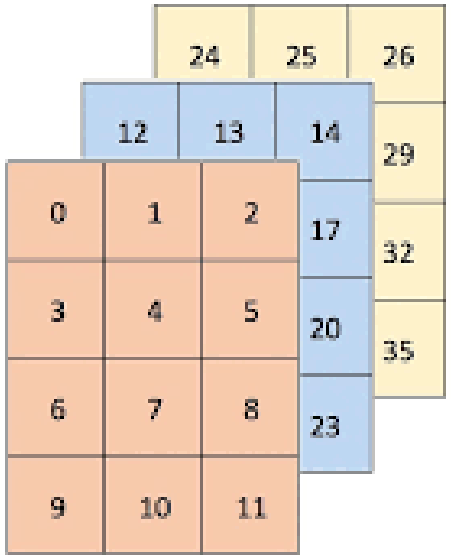
- Procesamiento eficiente de datos de sensores eléctricos
- Análisis de señales eléctricas (tensión, corriente, potencia)
- Procesamiento de datos de SCADA y sistemas de medición

PECIER
Comité Nacional Peruano de la CIER

Introducción a NumPy

La forma de utilizar esta biblioteca es:

```
import numpy as np
```



PECIER
Comité Nacional Peruano de la CIER

Creación de Arrays NumPy

Es común trabajar con grandes cantidades de datos, como mediciones de tensión, corriente, potencia, entre otros.

Para procesar esta información de forma eficiente, es recomendable utilizar estructuras optimizadas como los **arrays** de la biblioteca NumPy.

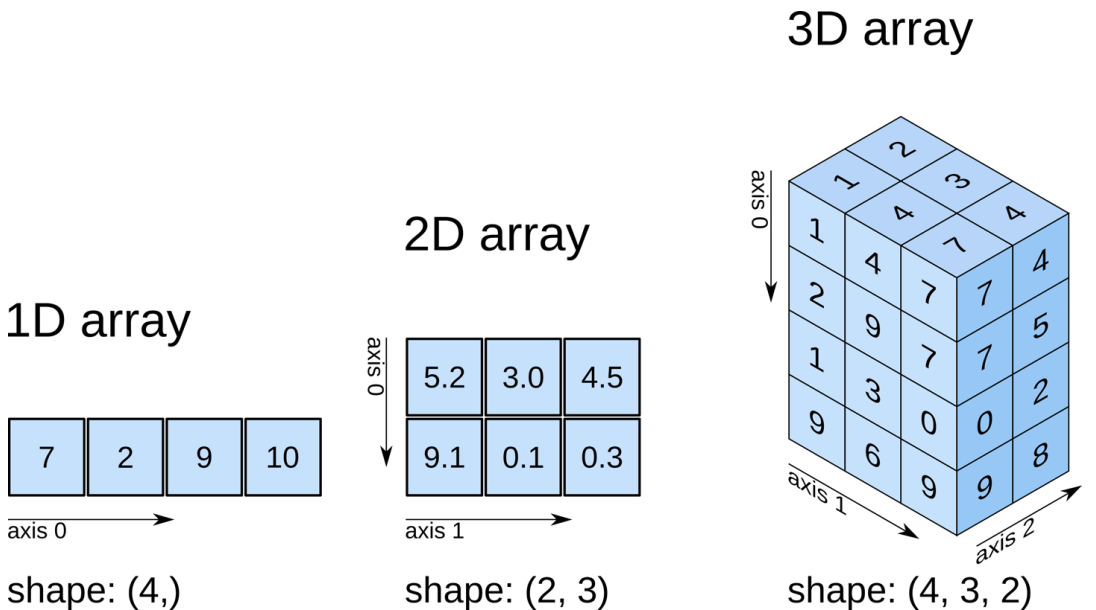
A diferencia de las listas de Python, los arrays de NumPy son más rápidos, ocupan menos memoria y permiten realizar operaciones vectoriales de forma sencilla.

PECIER
Comité Nacional Peruano de la CIER

Creación de Arrays NumPy

¿Qué es un Array?

Un **array** es una estructura de datos similar a una lista, pero todos sus elementos son del mismo tipo (por ejemplo, números flotantes) y están almacenados en memoria de forma contigua. Esto permite que operaciones matemáticas se realicen de manera más rápida y eficiente.



PECIER
Comité Nacional Peruano de la CIER

Creación de Arrays NumPy

Creación desde listas de Python

Una de las formas más simples de crear un array en NumPy es a partir de una lista de Python. Esto es útil cuando tenemos, por ejemplo, una serie de mediciones eléctricas tomadas en distintos momentos.

```
import numpy as np
tensiones = np.array([220.5, 221.3, 219.8, 220.2, 222.1])
# Mediciones de tensión en volts
```

PECIER
Comité Nacional Peruano de la CIER

Creación de Arrays NumPy

Arrays multidimensionales

También es posible crear **arrays multidimensionales**, lo cual es muy útil para representar sistemas eléctricos más complejos. Por ejemplo, en un sistema trifásico, cada fase tiene su propia serie de mediciones de tensión.

```
import numpy as np

sistema_trifasico = np.array([
    [220.1, 219.8, 220.5],  # Fase R
    [219.9, 220.2, 220.0],  # Fase S
    [220.3, 220.1, 219.7]   # Fase T
])
```

PECIER
Comité Nacional Peruano de la CIER

Creación de Arrays NumPy

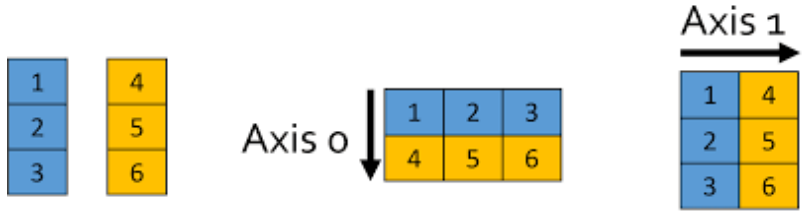
En este caso, estamos creando un array de dos dimensiones donde:

- Cada **fila** representa una **fase del sistema trifásico** (R, S y T).
- Cada **columna** representa una **medición tomada en un instante distinto del tiempo**.

PECIER
Comité Nacional Peruano de la CIER

Creación de Arrays NumPy

Este tipo de estructura permite realizar operaciones por fase, por instante de tiempo, o aplicar cálculos globales como promedios por fila o columna. Por ejemplo:




```
import numpy as np
# Promedio de tensión por fase
promedios_por_fase = np.mean(sistema_trifasico, axis=1)
# Promedio de tensión por instante de tiempo
promedios_por_instante = np.mean(sistema_trifasico, axis=0)
```

axis=0: calcula el promedio por columna (a lo largo de las filas).
axis=1: calcula el promedio por fila (a lo largo de las columnas).

PECIER
Comité Nacional Peruano de la CIER

Creación de Arrays NumPy

```
import numpy as np
# Promedio de tensión por fase
promedios_por_fase = np.mean(sistema_trifasico, axis=1)
# Promedio de tensión por instante de tiempo
promedios_por_instante = np.mean(sistema_trifasico, axis=0)
```

Entonces, estos arrays:

- Permiten representar fácilmente datos de sistemas monofásicos o trifásicos.
- Se integran con otras herramientas como gráficos, análisis estadísticos y simulaciones.
- Son ideales para automatizar el análisis de grandes volúmenes de datos provenientes de medidores inteligentes, SCADA o registros manuales.

PECIER
Comité Nacional Peruano de la CIER

Funciones especiales de Numpy

Numpy incluye una serie de funciones especiales que facilitan la creación de arrays sin necesidad de escribir manualmente cada valor.

Estas funciones son muy útiles para generar secuencias numéricas o estructuras comunes en el análisis eléctrico, como rangos de frecuencias o matrices identidad.

PECIER
Comité Nacional Peruano de la CIER

Funciones especiales de Numpy

`np.arange()`: Secuencias con paso definido

```
import numpy as np
# Rango de frecuencias de 50Hz a 60Hz con paso de 0.5Hz

frecuencias = np.arange(50, 60.5, 0.5)
```

Contexto práctico: En simulación de sistemas eléctricos puede ser necesario analizar el comportamiento del sistema a distintas frecuencias (u otra variable). Esta función genera un array que contiene frecuencias desde 50 Hz hasta 60 Hz (inclusive), con incrementos de 0.5 Hz. Así, podríamos simular o medir cómo varía un parámetro (como la corriente o la potencia) en ese rango. La utilidad es notoria conforme más sean los valores a probar.

PECIER
Comité Nacional Peruano de la CIER

Funciones especiales de Numpy

`np.linspace()`: Valores equidistantes entre dos extremos

```
import numpy as np
# 10 valores espaciados linealmente entre 0 y 10V
tensiones_lin = np.linspace(0, 10, 10)
```

Contexto práctico: Supongamos que queremos analizar cómo responde un circuito en una simulación ante variaciones de tensión desde 0V hasta 10V. La función `linspace()` genera 10 valores equidistantes entre esos dos extremos. Esto puede utilizarse, por ejemplo, para simular una rampa de tensión o para graficar una curva característica de un componente.

PECIER
Comité Nacional Peruano de la CIER

Funciones especiales de Numpy

`np.eye()`: Matriz identidad

```
import numpy as np # Matriz identidad 3x3 (como en matrices de impedancia)
matriz_identidad = np.eye(3)
```

Contexto práctico: En análisis matricial de sistemas eléctricos, especialmente en estudios de impedancia o admitancia (Z o Y), la **matriz identidad** juega un papel clave en cálculos algebraicos. Por ejemplo, al resolver sistemas de ecuaciones lineales de circuitos trifásicos balanceados, la matriz identidad puede representar la parte "neutra" del sistema.
La función `np.eye(n)` genera una matriz de tamaño n x n con unos en la diagonal principal y ceros en el resto, lo cual representa exactamente la matriz identidad.

PECIER
Comité Nacional Peruano de la CIER

Funciones especiales de Numpy

Comparación rápida entre funciones

Función	¿Qué hace?	Uso típico en ingeniería eléctrica
`np.arange()`	Crea un array con un paso fijo	Rango de frecuencias, tiempos de muestreo
`np.linspace()`	Divide un intervalo en partes iguales	Tensiones, divisiones de carga, escalas de gráfica
`np.eye()`	Crea una matriz identidad	Cálculos matriciales, transformadas, análisis Z/Y

PECIER
Comité Nacional Peruano de la CIER

Ejemplos y ejercicios

PECIER_dia4_parte1

colab.research.google.com

PECIER

Comité Nacional Peruano de la CIER

SESIÓN 4
Parte 2

PECIER
Comité Nacional Peruano de la CIER

Operaciones con Arrays

Uno de los grandes beneficios de utilizar Numpy en lugar de listas comunes de Python es la posibilidad de realizar operaciones matemáticas directamente sobre arrays, de forma **vectorizada**. Esto no solo simplifica el código, sino que también mejora el rendimiento y la claridad, especialmente útil al procesar datos eléctricos como tensiones, corrientes o potencias.

PECIER
Comité Nacional Peruano de la CIER

Operaciones con Arrays

Operaciones matemáticas básicas

Numpy permite realizar operaciones como suma, resta, multiplicación o división directamente entre arrays, elemento por elemento. Vimos un ejemplo:

```
import numpy as np
# Tensiones medidas en volts
tensiones = np.array([220.5, 221.3, 219.8, 220.2, 222.1])
# Corrientes medidas en amperes
corrientes = np.array([10.2, 9.8, 10.5, 11.1, 10.9])
# Potencias instantáneas calculadas como P = V * I
potencias = tensiones * corrientes
```

PECIER
Comité Nacional Peruano de la CIER

Operaciones con Arrays

Funciones matemáticas

NumPy también incluye funciones matemáticas comunes como sqrt(), mean(), sum(), max(), entre muchas otras. Estas se aplican a arrays completos y devuelven resultados muy útiles en análisis eléctrico.

```
import numpy as np
# Datos simulados tensiones = np.array([220.5, 221.3, 219.8, 220.2, 222.1])
# Calcular el valor RMS (Root Mean Square) de las tensiones
tensiones_rms = np.sqrt(np.mean(tensiones**2))
```

Contexto práctico: El valor **RMS** (valor eficaz) es fundamental en sistemas de corriente alterna (CA), ya que representa el valor equivalente de una señal AC en términos de potencia. Es decir, cuánto trabajo (energía) haría esa tensión si fuera una tensión continua equivalente.

PECIER
Comité Nacional Peruano de la CIER

Operaciones con Arrays

Indexación y slicing

Los arrays Numpy permiten seleccionar partes específicas de los datos mediante **indexación** (acceder a un solo elemento) o **slicing** (extraer subconjuntos).

```
# Obtener la primera fase del sistema trifásico
fase_R = sistema_trifasico[0, :]
```

- Aquí seleccionamos la **primera fila** del array sistema_trifasico, que corresponde a la fase **R**.
- El uso de : indica que queremos **todos los valores de esa fila**.

```
# Obtener valores impares del array de
tensiones tensiones_impares = tensiones[1::2]
```

- Este slicing selecciona desde el **índice 1** (el segundo elemento) hasta el final, **tomando de a dos elementos**.
- Útil para separar muestras alternadas, como por ejemplo lecturas tomadas en condiciones distintas (día/noche, carga ligera/pesada, etc.).

PECIER
Comité Nacional Peruano de la CIER

Operaciones con Arrays

Tabla de slicing en arrays NumPy

Supongamos que tenemos el siguiente array unidimensional:

```
arr = np.array([10, 20, 30, 40, 50, 60])
# Índices: 0 1 2 3 4 5
```

Expresión	Resultado	Descripción
`arr[:]`	`[10 20 30 40 50 60]`	Todo el array completo
`arr[0]`	`10`	Elemento en la posición 0
`arr[-1]`	`60`	Último elemento (índice negativo)
`arr[1:4]`	`[20 30 40]`	Desde índice 1 (incluido) hasta 4 (excluido)
`arr[:3]`	`[10 20 30]`	Desde el inicio hasta índice 3 (excluido)
`arr[3:]`	`[40 50 60]`	Desde índice 3 hasta el final
`arr[::2]`	`[10 30 50]`	Elementos con paso de 2 (pares por índice)
`arr[1::2]`	`[20 40 60]`	Elementos impares (por índice)
`arr[::-1]`	`[60 50 40 30 20 10]`	Array invertido (útil para análisis en reversa o señales de prueba)
`arr[2:5:2]`	`[30 50]`	Desde índice 2 hasta 5 (excluido), con paso de 2

PECIER
Comité Nacional Peruano de la CIER

Operaciones con Arrays

```
# Supongamos un array 2D:

mat = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
# Índices:
# Fila 0: [1 2 3]
# Fila 1: [4 5 6]
# Fila 2: [7 8 9]
```

Expresión	Resultado	Descripción
`mat[0]`	`[1 2 3]`	Primera fila
`mat[:, 0]`	`[1 4 7]`	Primera columna
`mat[1, :]`	`[4 5 6]`	Toda la segunda fila
`mat[1:3, 1:]`	`[[5 6] [8 9]]`	Submatriz desde fila 1 a 2, columna 1 a 2
`mat[::-1, :]`	`[[7 8 9] [4 5 6] [1 2 3]]`	Matriz invertida por filas
`mat[:, ::-1]`	`[[3 2 1] [6 5 4] [9 8 7]]`	Matriz invertida por columnas

PECIER
Comité Nacional Peruano de la CIER

Operaciones con Arrays

Aplicaciones típicas en sistemas eléctricos:

- Separar canales de un sistema trifásico (fase_R = sistema[0, :])
- Extraer muestras alternadas de una señal (senal_filtrada = senal[::2])
- Seleccionar subregiones para análisis (submatriz = datos[10:20, 0:3])

PECIER
Comité Nacional Peruano de la CIER

Ejemplo de aplicación

Estas operaciones permiten realizar análisis rápidos sin necesidad de estructuras de control complejas (como bucles). Por ejemplo:

```
# Potencia media consumida potencia_media = np.mean(potencias)
# Diferencia de tensión entre dos mediciones
delta_v = tensiones[4] - tensiones[0]
```

Este tipo de cálculos es esencial cuando se trabaja con grandes volúmenes de datos provenientes de sensores o registros históricos.

PECIER
Comité Nacional Peruano de la CIER

Álgebra Lineal Aplicada

En el análisis de circuitos eléctricos, especialmente cuando se trabaja con métodos sistemáticos como el **método de nodos** o el **método de mallas**, es muy común representar los problemas en forma de **sistemas de ecuaciones lineales**. Se puede programar su resolución con Python y Numpy.

PECIER
Comité Nacional Peruano de la CIER

Álgebra Lineal Aplicada

Representación matricial de circuitos eléctricos

Veamos un ejemplo sencillo basado en el **análisis nodal**, una técnica fundamental para resolver circuitos utilizando la Ley de Corrientes de Kirchhoff (LCK).

Supongamos que tenemos un circuito con dos nodos y se han obtenido las siguientes ecuaciones aplicando LCK:

$5V_1 - 2V_2 = 10$ $-2V_1 + 4V_2 = -5$ Este sistema representa las tensiones en dos nodos (V_1) y (V_2), en función de las resistencias conectadas y las fuentes de corriente/tensión del circuito.

PECIER
Comité Nacional Peruano de la CIER

Álgebra Lineal Aplicada

Podemos reescribirlo en forma matricial como:

$$\begin{bmatrix} 5 & -2 \\ -2 & 4 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 10 \\ -5 \end{bmatrix}$$

PECIER
Comité Nacional Peruano de la CIER

Álgebra Lineal Aplicada

Resolución con Numpy

Este tipo de sistema puede resolverse fácilmente con la función np.linalg.solve(), que calcula la solución exacta de un sistema lineal (Ax = b), si la matriz A es cuadrada e invertible.

```
import numpy as np

# Matriz de coeficientes (resistencias y conexiones
del circuito)
A = np.array([[5, -2],
              [-2, 4]])

# Vector de términos independientes (fuentes de
corriente/tensión)
b = np.array([10, -5])

# Cálculo de las tensiones en los nodos
voltajes_nodales = np.linalg.solve(A, b)
print("Voltajes en los nodos:", voltajes_nodales)
```

PECIER
Comité Nacional Peruano de la CIER

Álgebra Lineal Aplicada

¿Qué representa cada parte?

Elemento	Significado en el circuito
`A`	Matriz de conductancias o coeficientes (G o Y)
`b`	Vector de términos independientes (corrientes, fuentes)
`voltajes_nodales`	Solución: valores de tensión en cada nodo

Este tipo de análisis es la base del software de simulación como SPICE, donde se modelan miles de nodos utilizando álgebra lineal.

PECIER
Comité Nacional Peruano de la CIER

Álgebra Lineal Aplicada

Consideraciones prácticas

- `np.linalg.solve()` solo funciona si la matriz A es **cuadrada** (n×n) y **no singular** (es decir, tiene inversa).
- Si el sistema no tiene solución o tiene infinitas, NumPy lanzará un error (LinAlgError).

PECIER
Comité Nacional Peruano de la CIER

Ejemplos y ejercicios

PECIER_dia4_parte2

colab.research.google.com

PECIER

Comité Nacional Peruano de la CIER

SESIÓN 4
Parte 3

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

En sistemas eléctricos reales, las formas de onda de tensión y corriente no siempre son puramente senoidales. La presencia de **armónicos** (frecuencias múltiples de la fundamental) puede indicar problemas en la calidad de energía, afectando el desempeño de motores, transformadores y equipos electrónicos sensibles.

Análisis de Armónicos con FFT

Los **armónicos** son componentes de frecuencia superiores a la fundamental que se suman a la señal original. Por ejemplo, en una red de 60 Hz, el **tercer armónico** corresponde a una señal de 180 Hz. Podemos generar una señal sintética con una componente fundamental y un armónico, y luego analizarla con la

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

```
import numpy as np
import matplotlib.pyplot as plt

# Generar vector de tiempo: 1000 puntos en 0.1 segundos
t = np.linspace(0, 0.1, 1000)

# Frecuencia fundamental
f = 60  # Hz

# Señal con componente fundamental y tercer armónico
senal = 220 * np.sin(2 * np.pi * f * t) + 30 * np.sin(2 * np.pi * 3 * f * t)
```

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

- Esta señal simula una **tensión distorsionada**, con una componente principal de 220 V a 50 Hz (frecuencia de red) y un tercer armónico (150 Hz) con 30 V de amplitud.
- Es un escenario común en instalaciones industriales con cargas no lineales (variadores, UPS, equipos electrónicos).

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

```
# Aplicar FFT a la señal
fft_senal = np.fft.fft(senal)

# Calcular frecuencias asociadas
frecs = np.fft.fftfreq(len(senal), d=t[1] - t[0])
```

- `np.fft.fft()` transforma la señal del **dominio del tiempo** al **dominio de la frecuencia**, revelando las componentes armónicas.
- `np.fft.fftfreq()` genera el eje de frecuencias correspondiente.

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

Visualización del espectro de frecuencias

Podemos graficar el espectro para ver claramente qué frecuencias están presentes en la señal:

```
# Solo frecuencias positivas
positivas = frecs > 0
frecuencias = frecs[positivas]
magnitudes = np.abs(fft_senal)[positivas] / len(senal)

plt.figure(figsize=(10, 4))
plt.stem(frecuencias, magnitudes, basefmt=" ")
plt.title("Espectro de frecuencias - Análisis Armónico")
plt.xlabel("Frecuencia (Hz)")
plt.ylabel("Magnitud (V)")
plt.grid(True)
plt.show()
```

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

¿Qué información obtenemos?

- Se observan **picos** en 50 Hz y 150 Hz, indicando la presencia de la fundamental y el tercer armónico.
- Este análisis es muy similar al que realizan analizadores de calidad de energía o funciones FFT en osciloscopios digitales.

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

Aplicaciones en sistemas eléctricos

Aplicación	Descripción
Detección de armónicos	Identificación de fuentes de distorsión armónica
Mantenimiento predictivo	Detección temprana de fallos en motores y transformadores
Análisis de redes industriales	Evaluar el impacto de cargas no lineales (rectificadores, variadores, etc.)
Mejora de calidad de energía	Diseño de filtros pasivos o activos

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

Consideraciones técnicas

- En señales reales, la **resolución espectral** depende del número de muestras y la duración del muestreo.
- Para evitar errores de aliasing, se debe cumplir con el teorema de Nyquist (frecuencia de muestreo > 2× la frecuencia máxima esperada).
- En mediciones reales se recomienda aplicar una **ventana de Hann o Hamming**.

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

Cálculo del THD (Total Harmonic Distortion)

El **THD** mide cuánta distorsión armónica contiene una señal respecto a su componente fundamental. Es ampliamente utilizado en:

- Evaluación de distorsiones en tensiones o corrientes.
- Verificación de normas de calidad de energía.
- Diagnóstico de problemas causados por cargas no lineales.

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

Fórmula del THD

$$THD = \frac{\sqrt{V_2^2 + V_3^2 + V_4^2 \cdots}}{V_1}$$

Donde:

- (V_1) es la magnitud de la **frecuencia fundamental** (ej. 50 Hz).
- (V_n) son las magnitudes de los **armónicos** (150 Hz, 250 Hz, etc. en 50 Hz).

PECIER
Comité Nacional Peruano de la CIER

Procesamiento de Señales Eléctricas

Interpretación del THD

Rango de THD (%)	Interpretación
0–5%	Muy buena calidad (típico en redes limpias)
5–10%	Aceptable, pero se debe monitorear
>10%	Potencialmente problemático
>20%	Necesidad urgente de corrección

PECIER
Comité Nacional Peruano de la CIER

Estadística para Datos Eléctricos

Cuando se cuenta con grandes volúmenes de datos, por ejemplo, provenientes de sensores de tensión, corriente, frecuencia y potencia, se requiere aplicar técnicas estadísticas básicas para evaluar la estabilidad, detectar anomalías, verificar el cumplimiento de normas de calidad de energía, etc.

PECIER
Comité Nacional Peruano de la CIER

Estadística para Datos Eléctricos

Análisis de calidad de energía

Supongamos que tenemos mediciones horarias de tensión durante una semana (168 horas = 7 días × 24 h). Estas mediciones pueden provenir de un medidor inteligente o un sistema SCADA. Podemos simular estos datos con una distribución normal (gaussiana) centrada en 220 V, que es el valor típico nominal de baja tensión en muchos países.

```
import numpy as np # Simulación de 168 mediciones horarias de tensión (una semana)
tensiones_semana = np.random.normal(loc=220, scale=2.5, size=168)
```

- loc=220: define la **media esperada** de la tensión (valor nominal).
- scale=2.5: define la **desviación estándar**, que representa la variación típica.
- size=168: genera un punto por hora durante una semana completa.

PECIER
Comité Nacional Peruano de la CIER

Estadística para Datos Eléctricos

Cálculos estadísticos

Con estos datos, podemos obtener los valores estadísticos más importantes:

```
media = np.mean(tensiones_semana) # Valor promedio
desviacion = np.std(tensiones_semana) # Variabilidad de las mediciones
maximo = np.max(tensiones_semana) # Pico más alto de tensión
minimo = np.min(tensiones_semana) # Punto más bajo de tensión
```

PECIER
Comité Nacional Peruano de la CIER

Estadística para Datos Eléctricos

¿Qué nos dice cada métrica?

Métrica	Significado práctico en el sistema eléctrico
`media`	Valor medio de la tensión: útil para verificar si está dentro del rango permitido por normativa (ej. ±10%).
`desviacion`	Mide cuánto fluctúan las tensiones respecto al valor medio. Valores altos pueden indicar **inestabilidad**.
`máximo`	Punto más alto de la tensión, posible **sobrevoltaje momentáneo**.
`mínimo`	Punto más bajo registrado, posible **subtensión**.

PECIER
Comité Nacional Peruano de la CIER

Estadística para Datos Eléctricos

Visualización

Para reforzar el análisis, se puede graficar la serie de tensiones medidas y comparar con los límites aceptables:

```
import matplotlib.pyplot as plt

horas = np.arange(168)

plt.figure(figsize=(12, 4))
plt.plot(horas, tensiones_semana, label='Tensión (V)')
plt.axhline(230, color='green', linestyle='--', label='Valor nominal')
plt.axhline(230 * 0.9, color='red', linestyle=':', label='Límite inferior (207V)')
plt.axhline(230 * 1.1, color='red', linestyle=':', label='Límite superior (253V)')
plt.title("Tensión medida durante una semana")
plt.xlabel("Horas")
plt.ylabel("Tensión (V)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

PECIER
Comité Nacional Peruano de la CIER

Ejemplos y ejercicios

PECIER_dia4_parte3

colab.research.google.com

PECIER

Comité Nacional Peruano de la CIER

E-mail: pecier@cier.org