Laboratorio Intermedio **Name:** *Sergio Montoya Ramirez*
Tarea Estadistica *Angelica Lopez Duarte*
September 3, 2025

Nota

Durante el transcurso de este trabajo se hara uso de varios scripts en python (Particularmente para usar sympy que opinamos que es una de las maravillas del mundo moderno). Se pondra aqui el codigo utilizado sin embargo recomendamos fervientemente que lo mire en el repo de github: `https://github.com/demon-s1e7j/Universidad/tree/main/Semestre10/LabIntermedio/Tareas/Tarea2/code`. Este ya esta comodamente organizado para tener un proyecto de uv con el que manejar los requerimientos. Eh de confesar que no estan pensados para que alguien mas los ejecute y entienda lo que retorna pero creo que sigue siendo mucho mejor que simplemente quedarse con la implementacion mostrada en este documento.

1 Punto 2.1

Podemos usar el codigo

```python
from functools import wraps
from typing import Sequence, Union
import math as mt

Numeric = Union[int, float, complex]


def mean(data: Sequence[Numeric]) -> Numeric:
    return sum(data) / len(data)


def autofill_valmean(func):
    @wraps(func)
    def wrapper(
            data: Sequence[Numeric],
            *args,
            valmean: Numeric | None = None,
            **kwargs):
        if valmean is None:
            valmean = mean(data)
        return func(data, *args, valmean=valmean, **kwargs)
    return wrapper
```

```python
@autofill_valmean
def list_deviation(
        data: Sequence[Numeric],
        valmean: Numeric | None = None) -> Sequence[Numeric]:
    def deviation(x): return abs(valmean - x)
    return [deviation(x) for x in data]


@autofill_valmean
def standard_rough_and_ready(
        data: Sequence[Numeric],
        valmean: Numeric | None = None) -> Numeric:
    return (2 / 3) * (max(list_deviation(data, valmean=valmean)))


@autofill_valmean
def standard_2_3(
        data: Sequence[Numeric],
        valmean: Numeric | None = None) -> Numeric:
    return mt.sqrt(
        (sum(list_deviation(data, valmean=valmean))) / (len(data) - 1))


def standar_error(standard_deviation: Numeric, N: int) -> Numeric:
    return standard_deviation / mt.sqrt(N)


data = [25.8, 26.2, 26.0, 26.5, 25.8, 26.1, 25.8, 26.3]

if __name__ == "__main__":
    promedio = mean(data)
    desviacion_rar = standard_rough_and_ready(data)
    desviacion_2_3 = standard_2_3(data)
    error_rar = standar_error(desviacion_rar, len(data))
    error_2_3 = standar_error(desviacion_2_3, len(data))
    print(f"""\
    {promedio=}
    {desviacion_rar=}
    {desviacion_2_3=}
    {error_rar=}
    {error_2_3=}
    """)
```

lo que nos da:

```
$ uv run punto_2_1.py
```

```
promedio=26.0625
desviacion_rar=0.29166666666666663
desviacion_2_3=0.49280538030458104
error_rar=0.10311973892303816
error_2_3=0.17423301310929235
```

2 Punto 2.6

2.1

Numero de datos: 5

- $\alpha = 0.01913 \approx 0.019$
- $\bar{\delta} = 3.27346 \approx 3.273$

Resultado: 3.273 ± 0.019

2.2

Numero de datos: 50

- $\alpha = 0.002506 \approx 0.0025$
- $\bar{\delta} = 3.26513 \approx 3.2651$

Resultado: 3.25513 ± 0.019

2.3

Numero de datos: 500

- $\alpha = 0.000270 \approx 0.000270$
- $\bar{\delta} = 3.26681 \approx 3.26681$

Resultado: 3.273 ± 0.019

3 Punto 3.4

Podemos crear la funcion (sin integrarla) en sympy como

$$\frac{\sqrt{2}e^{-\frac{(-\mu+x)^2}{2\sigma^2}}}{2\sqrt{\pi}\sigma}$$

y con eso podemos implementar un codigo:

```python
import sympy
from tabulate import tabulate
import os

os.system("clear")


x, mean, std_dev = sympy.symbols('x mu sigma')

gaussian = (1 / (std_dev * sympy.sqrt(2 * sympy.pi))) * \
    sympy.exp(-((x - mean)**2) / (2 * std_dev**2))


def replicate_table_sympy(mean_val=0, std_dev_val=1):
    integral_sym = sympy.integrate(gaussian, x)

    ranges = [1, 1.65, 2, 2.58, 3, 4, 5]
    table_data = []

    for r in ranges:
        x_min = mean_val - r * std_dev_val
        x_max = mean_val + r * std_dev_val

        result = (integral_sym.subs(x, x_max) - integral_sym.subs(x, x_min)
        result = result.subs({mean: mean_val, std_dev: std_dev_val})

        fraction_in_range = float(result)

        fraction_out_range = 1 - fraction_in_range

        range_str = f"\\pm {r} \\sigma"
        in_range_percent = f"{fraction_in_range * 100:.2f}%"
        out_range_percent = f"{fraction_out_range * 100:.2f}%"
        table_data.append([range_str, in_range_percent, out_range_percent])

    headers = [
        "Centrado en media",
        "Medidas dentro del rango",
        "Medidas fuera del rango"]

    print(tabulate(table_data, headers=headers, tablefmt="fancy_grid"))
    print(tabulate(table_data, headers=headers, tablefmt="latex"))


replicate_table_sympy()
```

Que nos da como resultado:

Centrado en media	Medidas dentro del rango	Medidas fuera del rango
$\pm 1\sigma$	68.27%	31.73%
$\pm 1.65\sigma$	90.11%	9.89%
$\pm 2\sigma$	95.45%	4.55%
$\pm 2.58\sigma$	99.01%	0.99%
$\pm 3\sigma$	99.73%	0.27%
$\pm 4\sigma$	99.99%	0.01%
$\pm 5\sigma$	100.00%	0.00%

Que coincide con lo que esperamos

4 Punto 4.1

4.1 $Z = 2A$

- $\frac{dZ}{dA} = 2$
- $\delta Z = \left|\frac{dZ}{dA}\right|_{\bar{A}} \delta A = |2|(0.005) \approx 0.01$
- $Z \approx 18.54800 \pm 0.01000$

4.2 $Z = A/2$

- $\frac{dZ}{dA} = \frac{1}{2}$
- $\delta Z = \left|\frac{dZ}{dA}\right|_{\bar{A}} \delta A = |0.5|(0.005) \approx 0.0025$
- $Z \approx 4.63700 \pm 0.00250$

4.3 $Z = \frac{A-1}{A+1}$

- $\frac{dZ}{dA} = -\frac{A-1}{(A+1)^2} + \frac{1}{A+1}$
- $\delta Z = \left|\frac{dZ}{dA}\right|_{\bar{A}} \delta A = |0.018947|(0.005) \approx 9.4737e-05$
- $Z \approx 0.80533 \pm 0.00009$

4.4 $Z = \frac{A^2}{A-2}$

- $\frac{dZ}{dA} = -\frac{A^2}{(A-2)^2} + \frac{2A}{A-2}$
- $\delta Z = \left|\frac{dZ}{dA}\right|\_\bar{A}\delta A = |0.9244|(0.005) \approx 0.004622$
- $Z \approx 11.82390 \pm 0.00462$

4.5 $Z = \arcsin(\frac{1}{A})$

- $\frac{dZ}{dA} = -\frac{1}{A^2\sqrt{1-\frac{1}{A^2}}}$
- $\delta Z = \left|\frac{dZ}{dA}\right|\_\bar{A}\delta A = |-0.011695|(0.005) \approx 5.8476e-05$
- $Z \approx 0.10804 \pm 0.00006$

4.6 $Z = \sqrt{A}$

- $\frac{dZ}{dA} = \frac{1}{2\sqrt{A}}$
- $\delta Z = \left|\frac{dZ}{dA}\right|\_\bar{A}\delta A = |0.16419|(0.005) \approx 0.00082093$
- $Z \approx 3.04532 \pm 0.00082$

4.7 $Z = \ln(\frac{1}{\sqrt{A}})$

- $\frac{dZ}{dA} = -\frac{1}{2A}$
- $\delta Z = \left|\frac{dZ}{dA}\right|\_\bar{A}\delta A = |-0.053914|(0.005) \approx 0.00026957$
- $Z \approx -1.11361 \pm 0.00027$

4.8 $Z = \exp(A^2)$

- $\frac{dZ}{dA} = 2Ae^{A^2}$
- $\delta Z = \left|\frac{dZ}{dA}\right|\_\bar{A}\delta A = |4.1754e+38|(0.005) \approx 2.0877e+36$
- $Z \approx 2.251e+37 \pm 2.088e+36$

4.9 $Z = A + \sqrt{\frac{1}{A}}$

- $\frac{dZ}{dA} = 1 - \frac{\sqrt{\frac{1}{A}}}{2A}$
- $\delta Z = \left|\frac{dZ}{dA}\right|_{\bar{A}} \delta A = |0.9823|(0.005) \approx 0.0049115$
- $Z \approx 9.60237 \pm 0.00491$

4.10 $Z = 10^A$

- $\frac{dZ}{dA} = 10^A \log{(10)}$
- $\delta Z = \left|\frac{dZ}{dA}\right|_{\bar{A}} \delta A = |4.3273e+09|(0.005) \approx 2.1636e+07$
- $Z \approx 1.879e+09 \pm 2.164e+07$

5 Punto 4.4

Podemos reescribir la formula que nos pidieron en sympy y despejar la ecuacion 4.10 y con eso encontrar los resultados. Si lo hacemos para un valor generico (Es decir, θ_i y θ_t) los resultados son

$$R = \frac{\tan^2{(\theta_i - \theta_t)}}{\tan^2{(\theta_i + \theta_t)}}$$

$$\delta_R = \sqrt{\delta_{\theta_i}^2 \left(\frac{(2\tan^2{(\theta_i - \theta_t)} + 2)\tan{(\theta_i - \theta_t)}}{\tan^2{(\theta_i + \theta_t)}} + \frac{(-2\tan^2{(\theta_i + \theta_t)} - 2)\tan^2{(\theta_i - \theta_t)}}{\tan^3{(\theta_i + \theta_t)}}\right)^2 + \delta_{\theta_t}^2 \left(\frac{(-2\tan^2{(\theta_i - \theta_t)} - 2)\tan{(\theta_i - \theta_t)}}{\tan^2{(\theta_i + \theta_t)}} + \frac{(-2\tan^2{(\theta_i + \theta_t)} - 2)\tan^2{(\theta_i - \theta_t)}}{\tan^3{(\theta_i + \theta_t)}}\right)^2}$$

Ahora reemplazando a los valores que nos dieron para θ_i y θ_t como puede ver en el siguiente script:

```
import sympy as sp
import os

os.system("clear")

s_theta_i, s_theta_t, delta_theta_i, delta_theta_t = sp.symbols(
```

```
    "theta_i theta_t delta_{theta_i} delta_{theta_t}")


def R(theta_i=s_theta_i, theta_t=s_theta_t):
    num = sp.Pow(sp.tan(theta_i - theta_t), 2)
    den = sp.Pow(sp.tan(theta_i + theta_t), 2)
    return num / den


def dR_dA(R_expr, theta=s_theta_i):
    return sp.diff(R_expr, theta)


def delta_R(R_expr, e_theta_i, e_theta_t):
    dR_dtheta_i = dR_dA(R_expr, theta=s_theta_i)
    dR_dtheta_t = dR_dA(R_expr, theta=s_theta_t)
    return sp.sqrt((dR_dtheta_i * e_theta_i)**2 + (dR_dtheta_t * e_theta_t)


expresion = R()
error = delta_R(expresion, delta_theta_i, delta_theta_t)

print("-" * 60)
sp.print_latex(expresion)
print("\n")
sp.print_latex(error)
print("-" * 60)


def rad_of_grad(x): return (x * sp.pi) / 180


values = {
    s_theta_i: 45.0,
    delta_theta_i: 0.1,
    s_theta_t: 34.5,
    delta_theta_t: 0.2}
values = {key: rad_of_grad(value) for key, value in values.items()}

val_expresion = expresion.evalf(subs=values)
val_error = error.evalf(subs=values)

print(val_expresion)
print(val_error)
sp.print_latex(val_expresion)
```

```
sp.print_latex(val_error)
```

al ejecutar este script nos devuelve:

$$0.00118 \pm 9 \cdot 10^{-5}$$

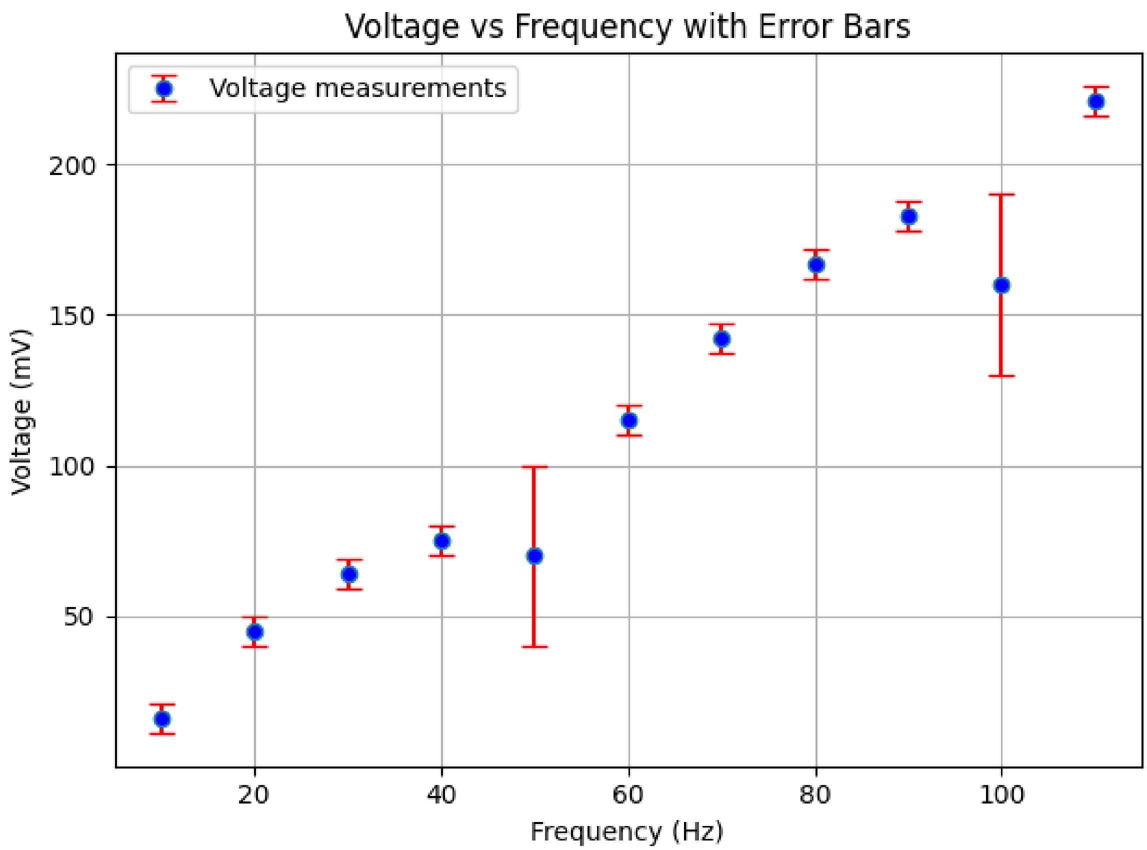
6 Punto 6.1

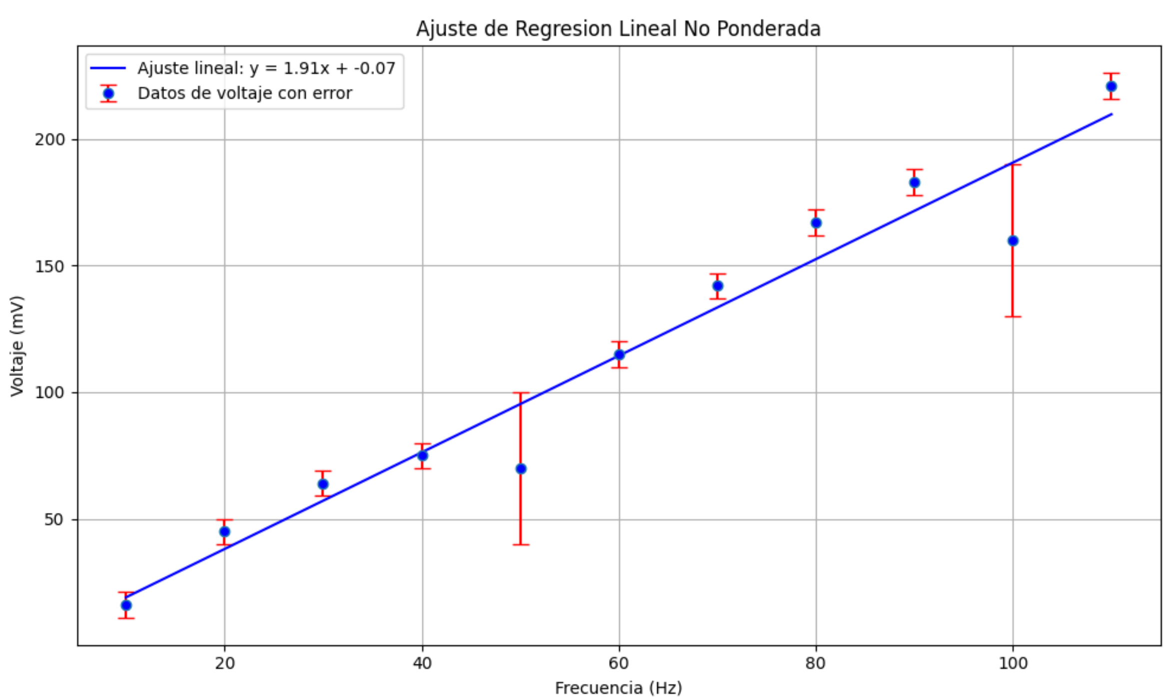
Para esto de nuevo usamos simplemente un script de python utilizando las ecuaciones 5.1 a 5.6 del libro. Las puede encontrar en el script como:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os

os.system("clear")

data = pd.read_csv("./data_6_1.csv")
print(data)

x, y, yerr = data["Frecuency (Hz)"].to_numpy(
), data["Voltage (mV)"].to_numpy(), data["Error (mV)"].to_numpy()

plt.errorbar(
    x,
    y,
    yerr=yerr,
    fmt='o',
    ecolor='red',
    capsize=5,
    linestyle='none',
    markerfacecolor='blue',
    label="Voltage measurements"
)

plt.xlabel("Frequency (Hz)")
plt.ylabel("Voltage (mV)")
plt.title("Voltage vs Frequency with Error Bars")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.savefig("punto_6_1_a.png")

sum_x, sum_y, sum_x_sq, sum_xy = (np.sum(x) for x in [x, y, x**2, x * y])
```

```
delta = len(x) * np.sum(x**2) - (np.sum(x))**2

num_c = (sum_x_sq * sum_y) - (sum_x * sum_xy)
c = num_c / delta

num_m = (len(x) * sum_xy) - (sum_x * sum_y)
m = num_m / delta

alpha_CU = np.sqrt(1 / (len(x) - 2) * np.sum((y - m * x - c)**2))

alpha_c = alpha_CU * np.sqrt(sum_x_sq / delta)
alpha_m = alpha_CU * np.sqrt(len(x) / delta)

plt.figure(figsize=(10, 6))

plt.errorbar(
    x,
    y,
    yerr=yerr,
    fmt='o',
    ecolor='red',
    capsize=5,
    linestyle='none',
    markerfacecolor='blue',
    label="Datos de voltaje con error"
)

plt.plot(
    x,
    m * x + c,
    color='b',
    label=f'Ajuste lineal: y = {m:.2f}x + {c:.2f}'
)

plt.xlabel("Frecuencia (Hz)")
plt.ylabel("Voltaje (mV)")
plt.title("Ajuste de Regresion Lineal No Ponderada")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.savefig("punto_6_1_b.png")
```

Y esto nos da como respuesta


Voltage vs Frequency with Error Bars
Voltage measurements
Voltage (mV)
Frequency (Hz)
50
100
150
200
20
40
60
80
100


Ajuste de Regresion Lineal No Ponderada
Ajuste lineal: y = 1.91x + -0.07
Datos de voltaje con error
Voltaje (mV)
Frecuencia (Hz)
50
100
150
200
20
40
60
80
100