**Documentación Técnica: Experimento 1.1**

#### **1. Introducción**

El presente informe detalla un análisis avanzado de señales de vibración capturadas mediante tres tipos de sensores (plano, circular y zumbador) en diferentes distancias, utilizando grabaciones de audio procesadas (.wav). El objetivo principal es caracterizar las señales obtenidas y desarrollar modelos predictivos basados en un ajuste exponencial y en redes neuronales, con el fin de estimar parámetros clave como la energía inicial transferida y el coeficiente de atenuación del medio. Este enfoque combina técnicas de preprocesamiento, cálculo energético y modelado estadístico para obtener resultados precisos y representativos de la propagación de vibraciones en un entorno controlado.

#### **2 Preparación y Carga de los Datos**

##### **2.1 Archivos Procesados**

Los datos procesados consisten en archivos de audio (.wav) con registros a diferentes distancias y con tres tipos de sensores:

* **Sensor Plano**: Archivos test1.wav, test2.wav, test3.wav.
* **Sensor Circular**: Archivos test1.wav, test2.wav, test3.wav.
* **Sensor Zumbador**: Archivos test1.wav, test2.wav, test3.wav.

Cada archivo se analiza independientemente para identificar patrones característicos.

##### **2.2 Preprocesamiento de las Señales**

El código utilizado para cargar y normalizar las señales es el siguiente:

from scipy.io import wavfile

import numpy as np

def load\_and\_preprocess(signal\_data):

"""

Carga un archivo de audio y realiza el preprocesamiento:

- Normalización de la amplitud.

- Cálculo del tiempo correspondiente a cada muestra.

"""

sample\_rate, data = wavfile.read(signal\_data)

if len(data.shape) > 1:

data = np.mean(data, axis=1) # Promedio si el audio tiene más de un canal

normalized\_data = data / np.max(np.abs(data)) # Normalización

time = np.arange(0, len(data)) / sample\_rate # Vector de tiempo

return sample\_rate, time, normalized\_data

Este método garantiza que las señales tengan una escala uniforme, independientemente de la intensidad original.

#### **3 Análisis de la vibración completa**

En este apartado se tomarán las vibraciones completas, incluyendo rebotes para cada grabación de los tres sensores. Para poder comparar las grabaciones de cada sensor, es necesario agruparlas por intervalos ya que cada vibración de una misma grabación pertenece a una distancia distinta. De esa forma, para cada ascensor se tendrán tres medidas por cada distancia y se podrá evaluar tanto el modelo exponencial como entrenar un modelo basado en una red neuronal.

##### **3.1 Definición de intervalos**

Para separar cada grabación en los intervalos correctos y analizar cada vibración por separado:

def define\_intervals(manual\_intervals, time):

intervals = []

for start\_time, end\_time in manual\_intervals:

start\_idx = np.searchsorted(time, start\_time)

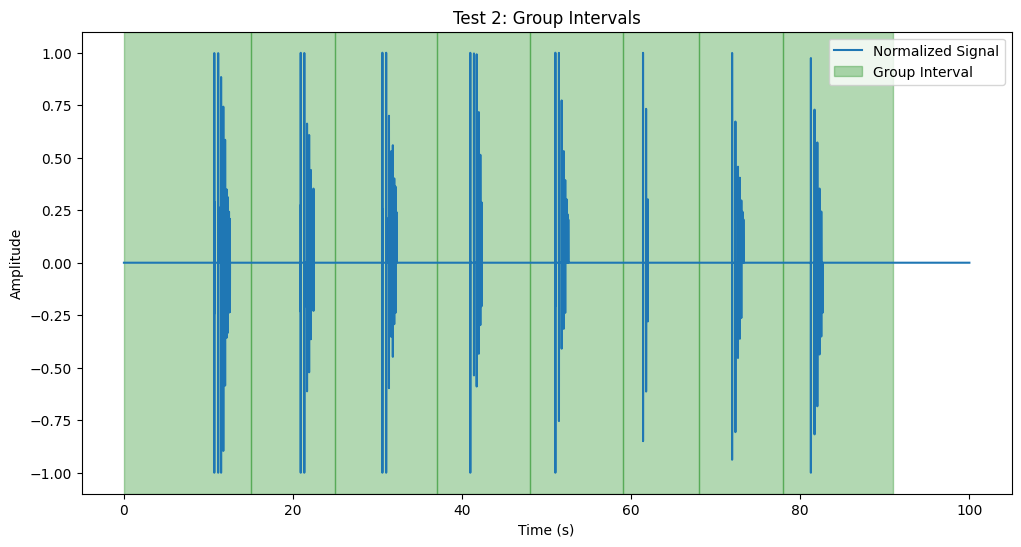
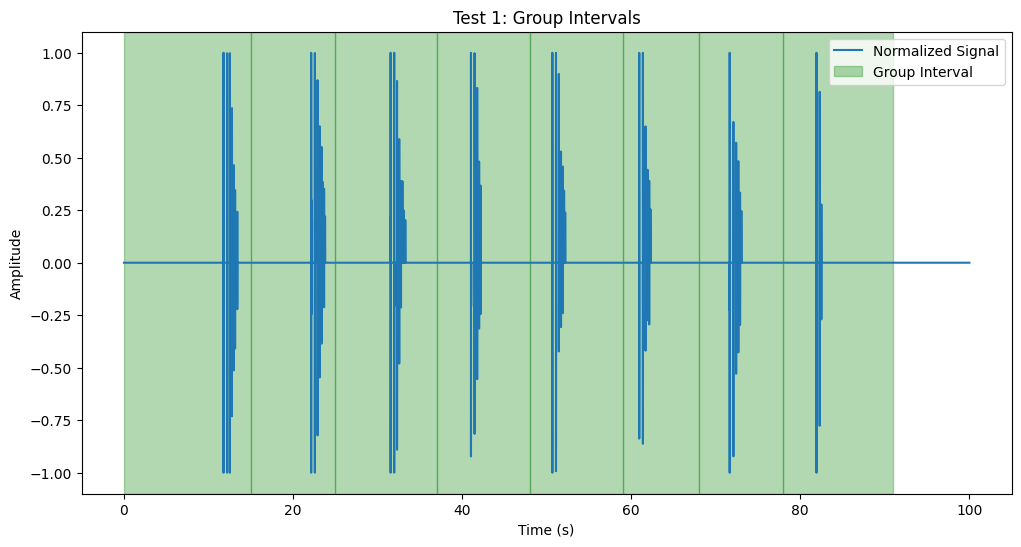
end\_idx = np.searchsorted(time, end\_time)

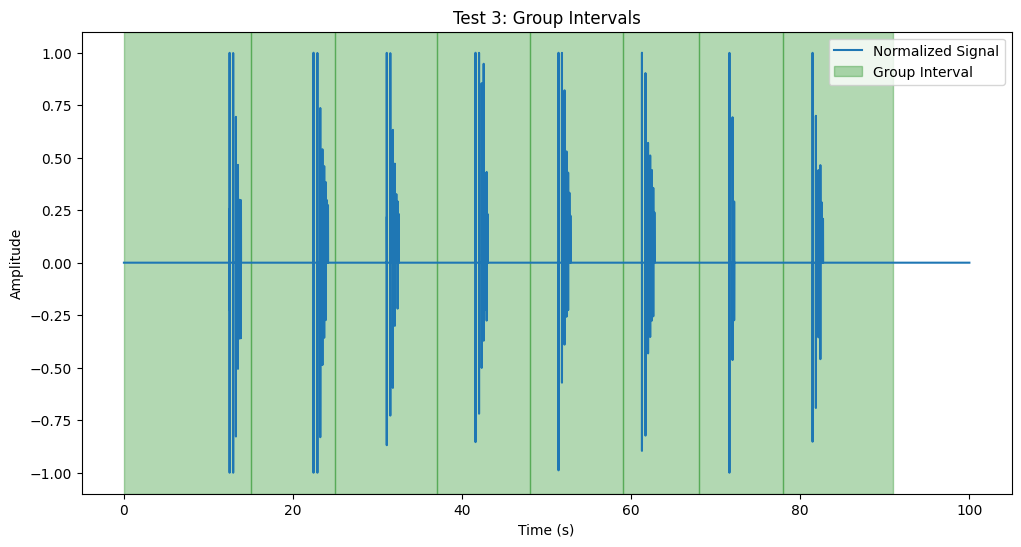
intervals.append((start\_idx, end\_idx))

return intervals

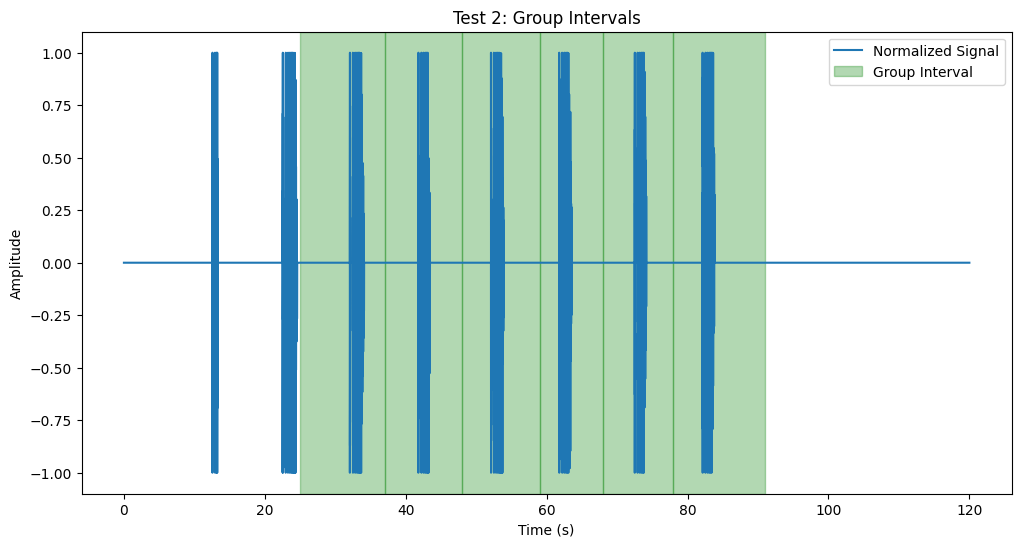
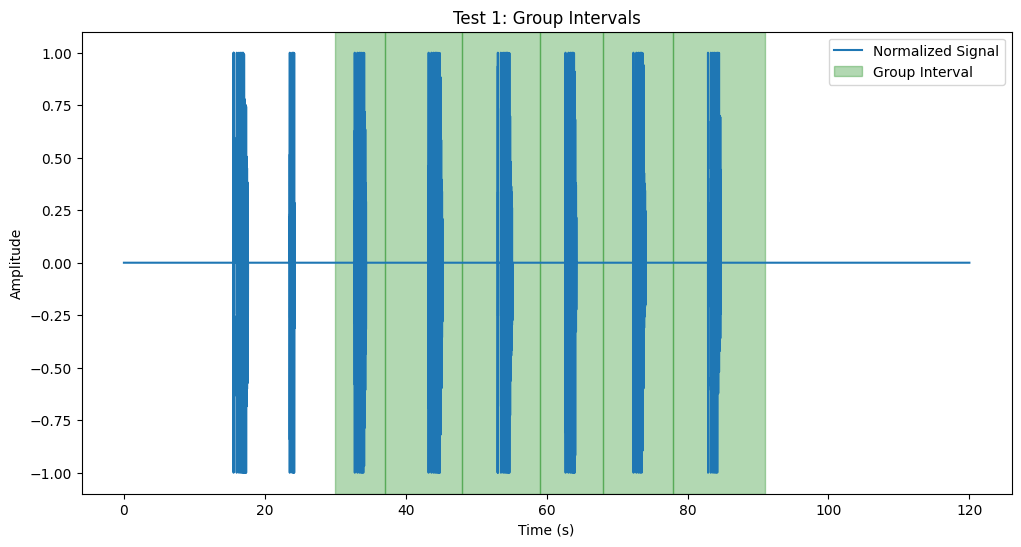
Al tener tres grabaciones para cada uno de los tres sensores, se realiza este proceso 9 veces obteniendo las siguientes gráficas:

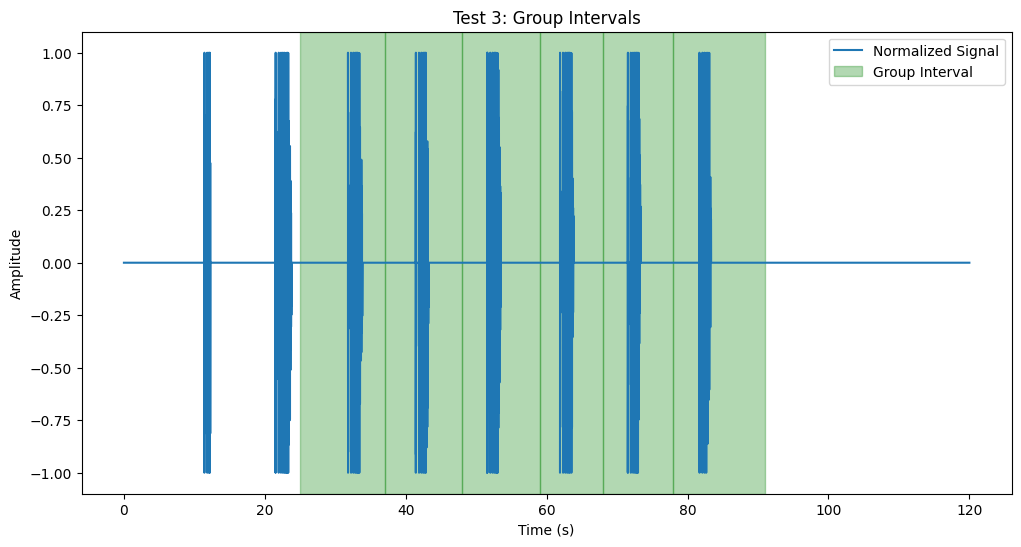
* Sensor plano:



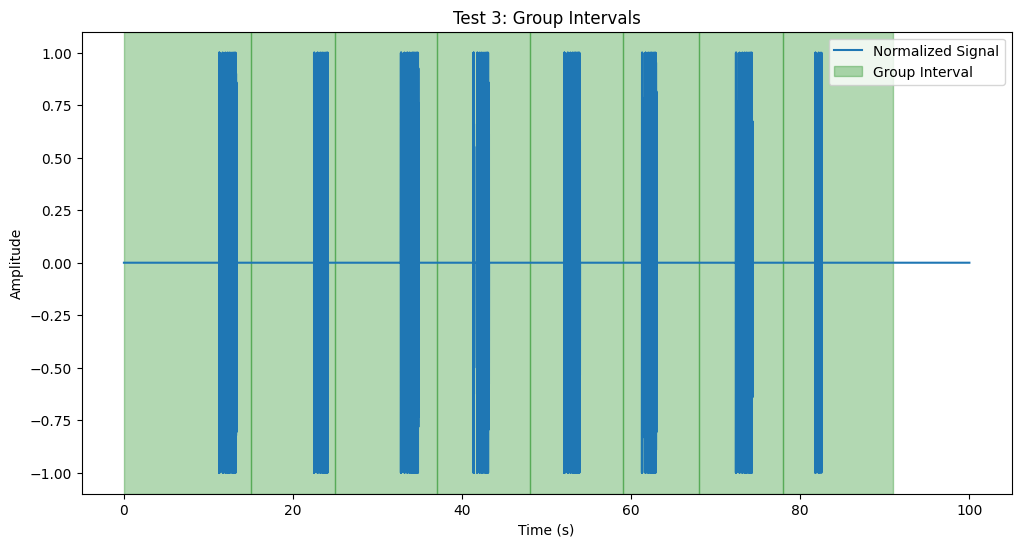
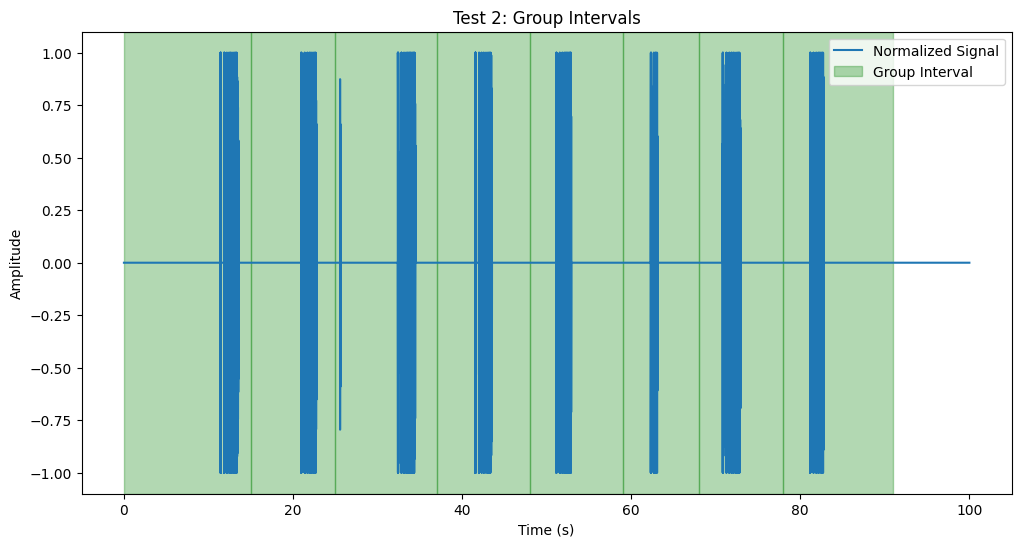
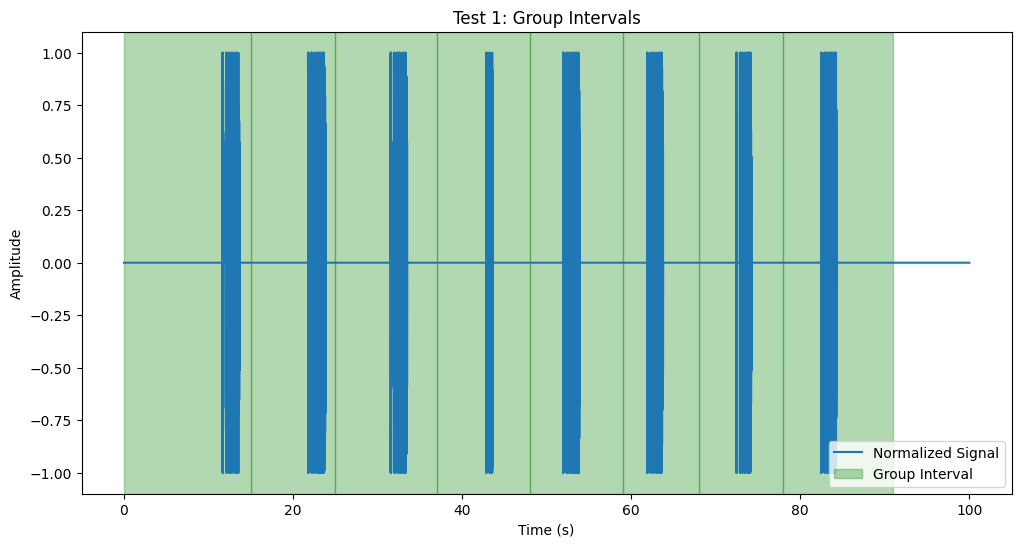


* Sensor plano circular





* Sensor zumbador



##### **3.2 Cálculo de la energía**

Se calcula energía para cad auna de las vibraciones asociadas a una distancia por el método de Simpson:

def calculate\_group\_energy(normalized\_data, interval, sample\_rate):

start\_idx, end\_idx = interval

segment = normalized\_data[start\_idx:end\_idx] # Include the entire group

energy = simpson(np.abs(segment)\*\*2, dx=1/sample\_rate) # Use absolute values for integration

return energy

Las energías calculadas se guardan en un dataframe junto a la distancia a la que han sido medidas.

* Sensor plano

Distance Amplitude Test 1 Amplitude Test 2 Amplitude Test 3

0 120 0.002117 0.002173 0.001503

1 150 0.001706 0.001509 0.001511

2 180 0.001491 0.001667 0.001033

3 210 0.001201 0.001337 0.001323

4 240 0.001229 0.001090 0.000849

5 270 0.001015 0.000691 0.001189

6 300 0.001341 0.000954 0.000692

7 330 0.000822 0.001164 0.000737

* Sensor plano circular

Distance Amplitude Test 1 Amplitude Test 2 Amplitude Test 3

0 180 0.020732 0.030656 0.029627

1 210 0.027147 0.024086 0.024394

2 240 0.032956 0.025366 0.030043

3 270 0.019995 0.022855 0.025608

4 300 0.019347 0.022848 0.027339

5 330 0.024885 0.028200 0.019781

* Sensor zumbador

Distance Amplitude Test 1 Amplitude Test 2 Amplitude Test 3

0 120 0.108426 0.118146 0.121934

1 150 0.111776 0.098023 0.089946

2 180 0.096940 0.116597 0.112039

3 210 0.037550 0.100007 0.089990

4 240 0.112158 0.113224 0.116089

5 270 0.087843 0.055464 0.088908

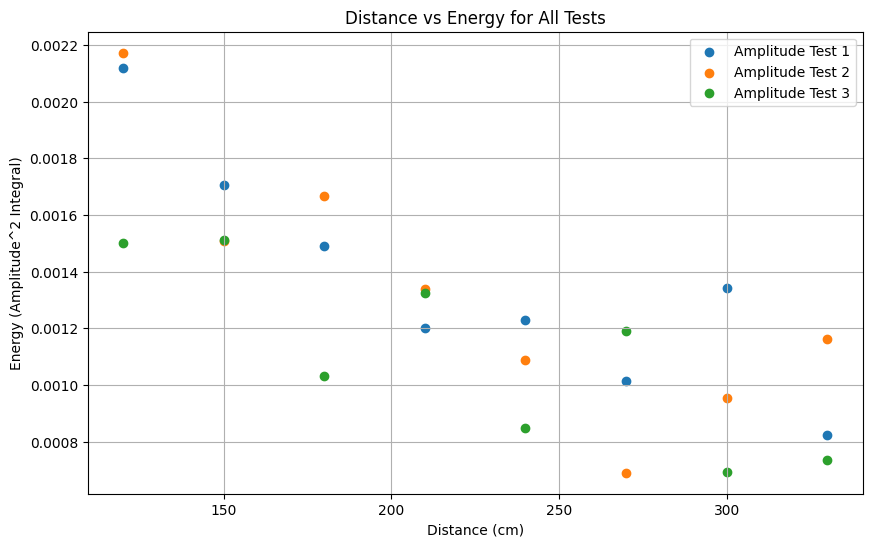
6 300 0.091547 0.118159 0.112816

7 330 0.092588 0.085823 0.049342

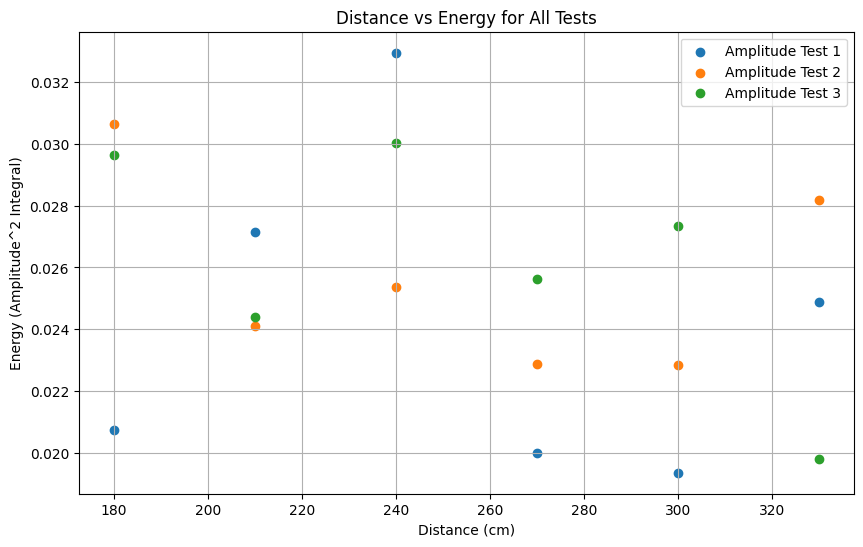
##### **3.3 Visualización de resultados**

A partir de los datos de los data-frames se obtienen las siguientes figuras:

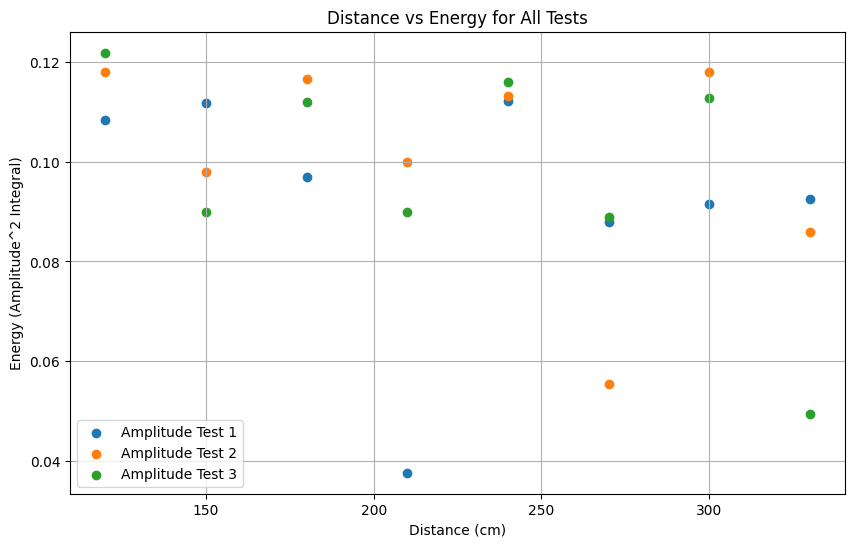
* Sensor plano



* Sensor plano circular



* Sensor zumbador



##### **3.4 Creación del modelo exponencial**

Se ajusta un modelo exponencial a los datos de los 3 data-frames. La ecuación que rige el modelo exponencial es:

Donde E0 es la energía transferida por la pelota al suelo al impactar con el y alpha es el coeficiente de atenuación del medio. En base a los datos de energía calculados y la distancia al que corresponden, se puede elaborar un modelo exponencial en el que se obtenga el valor de la energía “inicial” y el coeficiente de atenuación. Si el ajuste del modelo tiene una correlación alta con la realidad, se podría estimar el coeficiente de atenuación y por tanto poder desarrollar un modelo de cálculo de masas en base a los impactos sabiendo cuánto se reduce la energía recibida del sensor en comparación con la realmente transferida por el objeto.

# Función para ajustar el modelo exponencial

def fit\_exponential\_model(df):

"""

Ajusta un modelo exponencial a los datos del DataFrame.

"""

distances = df['Distance'].values

energies = df.iloc[:, 1:].mean(axis=1).values # Promedio de las energías

# Ajustar el modelo exponencial

def exponential\_decay(d, E0, alpha):

return E0 \* np.exp(-alpha \* d)

popt, \_ = curve\_fit(exponential\_decay, distances, energies, p0=(energies[0], 0.01))

E0, alpha = popt

# Predicciones del modelo exponencial

predicted\_energies = exponential\_decay(distances, E0, alpha)

# Calcular R^2

r2\_exponential = r2\_score(energies, predicted\_energies)

return E0, alpha, r2\_exponential, predicted\_energies

##### **3.5 Creación del modelo basado en redes neuronales**

En base a los datos de los 3 data-framas se puede entrenar un modelo de red neuronal que prediga la energía medida a diferentes distancias. Para ello, se aplica la siguiente metodología:

* Datos de Entrada y Salida
  + Variable de Entrada: Distancia (XXX), extraída de una columna específica de un conjunto de datos.
  + Variable Objetivo: Promedio de energías (yyy), calculado a partir de las columnas restantes del conjunto de datos.
* Preprocesamiento
  + Los datos fueron normalizados al rango [0, 1] utilizando la técnica de escalado MinMax, garantizando una escala homogénea entre las variables para optimizar el rendimiento del modelo.
* División del Conjunto de Datos
  + Se separaron los datos en un conjunto de entrenamiento (80%) y uno de prueba (20%), asegurando la validación independiente del modelo.
* Arquitectura del Modelo La red neuronal fue diseñada con las siguientes características:
  + Capa de Entrada y Oculta 1: 64 neuronas, activación ReLU.
  + Capa Oculta 2: 32 neuronas, activación ReLU.
  + Capa de Salida: 1 neurona, activación lineal. Esta arquitectura busca capturar patrones complejos en la relación distancia-energía.
* Entrenamiento
  + Se utilizó el optimizador Adam y la función de pérdida de error cuadrático medio (MSE), entrenando durante 500 épocas con un tamaño de lote de 8.
  + Los datos de entrenamiento se ajustaron iterativamente para minimizar el error entre predicciones y valores reales.
* Predicción y Evaluación
  + Se realizaron predicciones para todas las distancias utilizando el modelo entrenado.
  + Las predicciones fueron desescaladas al rango original de las energías.
  + El desempeño se evaluó mediante el coeficiente de determinación (R^2), el cual mide lo bien o mal que el modelo explica la variabilidad observada en los datos.

# Función para entrenar el modelo de red neuronal

def train\_neural\_network(df):

"""

Entrena una red neuronal sobre los datos del DataFrame.

"""

X = df[['Distance']].values

y = df.iloc[:, 1:].mean(axis=1).values.reshape(-1, 1) # Use the mean of all tests as target

# Escalar datos

scaler\_X = MinMaxScaler()

scaler\_y = MinMaxScaler()

X\_scaled = scaler\_X.fit\_transform(X)

y\_scaled = scaler\_y.fit\_transform(y)

# Dividir en entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_scaled, test\_size=0.2, random\_state=42)

# Crear y entrenar la red neuronal

model = Sequential([

Dense(64, activation='relu', input\_dim=1),

Dense(32, activation='relu'),

Dense(1, activation='linear')

])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

model.fit(X\_train, y\_train, epochs=500, batch\_size=8, verbose=0)

# Predicciones de la red neuronal en el conjunto completo

all\_distances\_scaled = scaler\_X.transform(X)

predicted\_energies\_scaled = model.predict(all\_distances\_scaled)

predicted\_energies = scaler\_y.inverse\_transform(predicted\_energies\_scaled).flatten()

# Calcular R^2

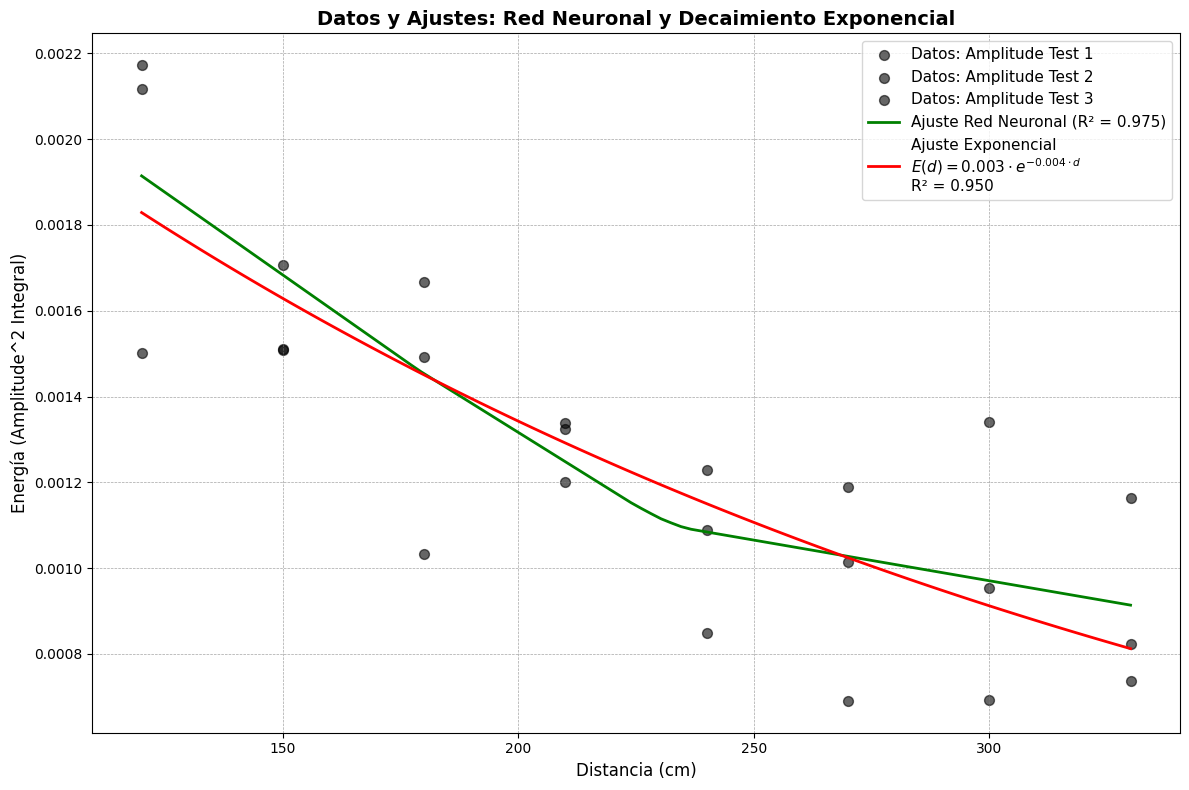
r2\_nn = r2\_score(y, predicted\_energies)

return model, scaler\_X, scaler\_y, r2\_nn, predicted\_energies

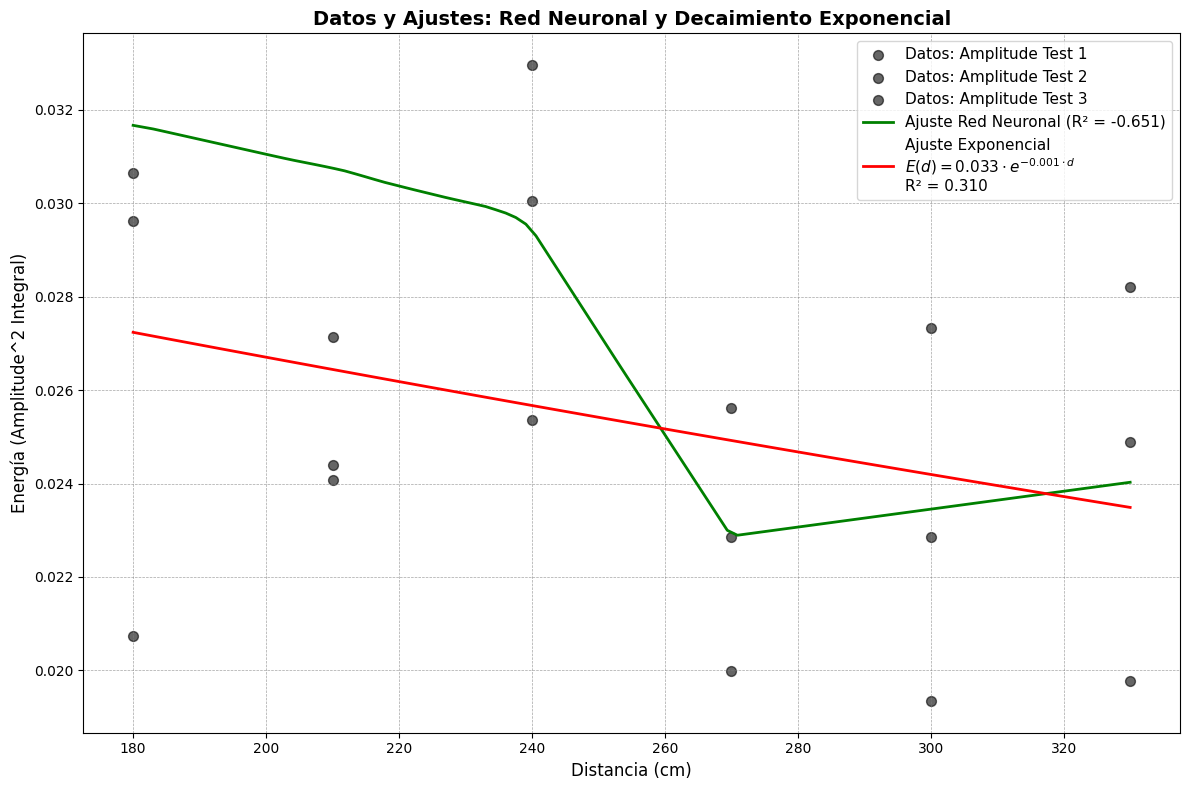
##### **3.6 Visualización de resultados**

Una vez se han elaborado los dos modelos, se aplican sobre los datos y se calcula su R^2 para evaluar su correlación con los datos reales.

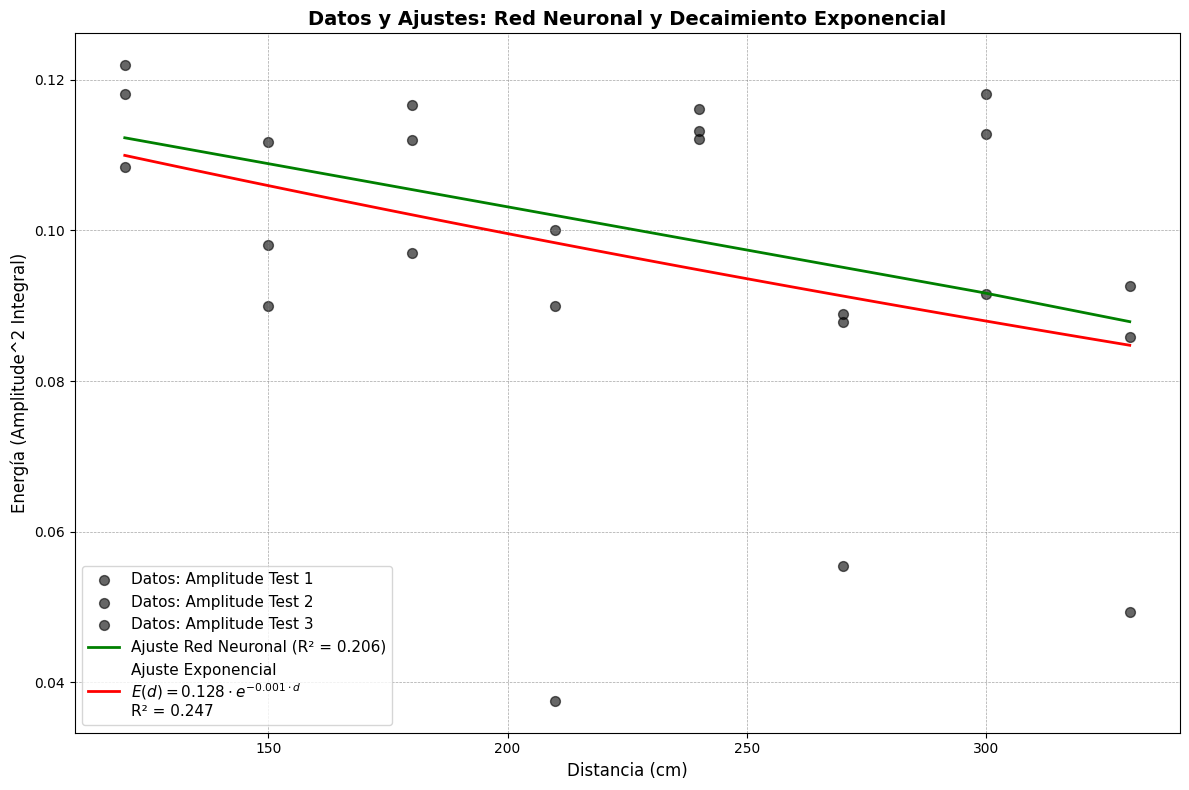
* Sensor plano

****

* Sensor plano circular

****

* Sensor zumbador

****

Como se puede observar, hay claras diferencias entre los sensores. Tanto el sensor plano circular como el zumbador presentan datos muy extremos debido a la saturación en la toma de datos. Sin embargo, el sensor plano presenta unos datos de energía acordes a ambos modelos.

Centrando el análisis en este último sensor, se puede ver que el modelo exponencial se ajusta adecuadamente proporcionando un valor de energía inicial y constante de atenuación lógicos. Además la correlación del modelo con los datos es muy alta 0.95 que podría mejorarse aún más con la toma de más datos.

Además, analizando la energía inicial obtenida y desnormalizado su valor, se puede observar que el valor obtenido es en torno a un 4.5% de la energía total que tendrían las pelotas de petanca al impactar en el suelo. Esto es un valor razonable, sabiendo que el material del suelo es hormigón y que solo deja propagar entre el 1 y el 10% de la energía transmitida.

En cuanto a la constante de atenuación, se obtiene un valor adecuado al material del suelo, ya que para vibraciones medias se encuentra entre el 0.003 y el 0.005. El valor obtenido es de 0.0039 lo que entra dentro del rango anterior.