UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Estimación Matemática y Computacional de Individuos Similares a un Patrón: Caso "Fedelobo"

PRESENTA

Alexander Eduardo Rojas Garay

Investigador, Matemático y Físico, UNAM

PROYECTO

Simulación de Parecidos Faciales

FECHA

17 de mayo de 2025

Índice

1. Introducción

La identificación de similitudes faciales en extensos conjuntos de datos posee aplicaciones en biometría, seguridad, personalización de contenido y estudios demográficos. Los algoritmos de aprendizaje profundo han demostrado alta efectividad, pero su complejidad y naturaleza de "caja negra" dificulta la interpretación y validación de sus resultados.

En este trabajo proponemos un enfoque basado en estadística clásica que combina:

- Reducción de dimensionalidad mediante Análisis de Componentes Principales (PCA).
- Métrica de similitud con distancia de Mahalanobis considerando la covarianza de los rasgos.
- Definición de umbrales basados en la distribución chi-cuadrada para clasificar parecidos.
- Implementación reproducible en Python y despliegue interactivo con Streamlit.

Este método permite:

1. Mantener transparencia en el pipeline de análisis.
2. Auditar cada paso desde datos crudos hasta resultados finales.
3. Ajustar dinámicamente parámetros de corte y explorarlos visualmente.

2. Objetivos

- Generar una muestra sintética de rasgos faciales basada en distribuciones normal multivariada.
- Normalizar y limpiar los datos, gestionando valores faltantes y atípicos.
- Aplicar PCA para reducir ruido y mejorar estabilidad numérica.
- Calcular distancias de Mahalanobis en el espacio reducido.
- Determinar umbrales de similitud con χ^2 y evaluar su impacto.
- Desplegar un dashboard para visualizar parecidos y mapas de México.

3. Revisión de Literatura

El uso de la distancia de Mahalanobis, propuesta por Mahalanobis en 1936, incorpora la estructura de covarianza de los datos, permitiendo medir similitud multidimensional. De Maesschalck et al. (2000) demostraron su aplicación en detección de objetos raros y validación de patrones biométricos.

Por otro lado, Jolliffe (2002) formalizó el Análisis de Componentes Principales (PCA) para reducir dimensionalidad maximizando la varianza. El enfoque PCA+Mahalanobis ha sido utilizado en:

- Clasificación de expresiones faciales (Wang et al., 2018).
- Estudios morfométricos en poblaciones humanas (Li et al., 2021).
- Detección de anomalías en series temporales y datos industriales.

Sin embargo, la integración de estas técnicas con dashboards ligeros (Streamlit, Plotly, Pydeck) es un aporte práctico que facilita la exploración y validación de resultados por usuarios no técnicos.

4. Estructura del Documento

Este reporte se organiza en cinco grandes bloques:

1. Introducción, objetivos y revisión de literatura.
2. Descripción de los datos sintéticos y verificación estadística.
3. Preprocesamiento, limpieza, estandarización e ingeniería de características.
4. PCA y modelo de distancia de Mahalanobis, con determinación de umbrales.
5. Resultados, análisis de sensibilidad, discusión, conclusiones y referencias.

5. Descripción de los Datos Sintéticos

La base de datos sintética consta de $N = 10\,000$ observaciones, cada una con cuatro variables continuas:

- **face_ratio**: Relación ancho/alto del rostro.
- **eye_height**: Altura relativa de los ojos al rostro.
- **eye_distance**: Distancia interpupilar normalizada.
- **brow_thickness**: Grosor medio de las cejas.

Estos rasgos se generaron a partir de una distribución normal multivariada:

$$\boldsymbol{\mu} = [1, 1, 1, 1], \quad \Sigma = \mathrm{diag}([0.01, 0.0064, 0.0144, 0.0025]).$$

La matiz diagonal simula varianzas extraídas de estudios antropométricos en población mexicana (Morales et al., 2015), garantizando realismo en la muestra.

```python
# data_generation.py

import numpy as np
import pandas as pd

def generate_synthetic_data(n_samples=10_000, seed=42):
    """
    Genera una muestra sint tica de rasgos faciales basada en distribuci n normal
        multivariada.
    Par metros
    ----------
    n_samples : int
        N mero de observaciones a generar.
    seed : int
        Semilla para reproducibilidad.
    Retorna
    -------
    df : pandas.DataFrame
        DataFrame con columnas ['face_ratio','eye_height','eye_distance','
            brow_thickness'].
    """
    np.random.seed(seed)
```

6. Preprocesamiento de Datos

Antes de aplicar el modelo, es imprescindible limpiar, validar y escalar las variables para garantizar coherencia y calidad en los resultados.

6.1. Carga y Exploración Inicial

Se inicia cargando la base de datos generada y realizando un análisis exploratorio inicial.

```
import pandas as pd

# Carga del archivo CSV
df = pd.read_csv("fedelobo_simulacion.csv")

# Vista de las primeras filas
print(df.head())

# Descripci\'on estad\'istica inicial
print(df.describe())
```

6.2. Imputación de Valores Faltantes

Verificamos la presencia de valores nulos y, de existir, se imputan con la mediana para no sesgar la distribución.

```
# Conteo de valores faltantes por columna
missing = df.isnull().sum()
print("Valores faltantes:\n", missing)

# Imputar nulos con mediana
df.fillna(df.median(), inplace=True)

# Confirmar que ya no hay nulos
print("Valores faltantes tras imputaci\'on:\n", df.isnull().sum())
```

6.3. Detección y Remoción de Atípicos

Se identifican outliers mediante puntuaciones Z y se eliminan aquellos con —z— ¿ 3.

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}, \quad |z_{ij}| > 3 \;\Rightarrow\; \text{outlier}$$

```
from scipy import stats
import numpy as np

# C\'alculo de z-scores
z_scores = stats.zscore(df)
abs_z = np.abs(z_scores)

# Mascara de filas con al menos un z-score fuera de [-3,3]
mask_outliers = (abs_z > 3).any(axis=1)
n_outliers = mask_outliers.sum()
print(f"Outliers detectados: {n_outliers}")

# Eliminar outliers
df_clean = df.loc[~mask_outliers].reset_index(drop=True)
print(f"Observaciones tras limpieza: {len(df_clean)}")
```

6.4. Ingeniería de Características

Se añade una variable derivada que puede proveer información adicional sobre proporciones faciales.

$$\text{eye\_brow\_ratio} = \frac{\text{eye\_height}}{\text{brow\_thickness}}.$$

```
# Crear variable derivada
df_clean["eye_brow_ratio"] = (
    df_clean["eye_height"] / df_clean["brow_thickness"]
)

# Ver primeras filas con la nueva variable
print(df_clean[["eye_height","brow_thickness","eye_brow_ratio"]].head())
```

6.5. Escalado de Variables

Para homogenizar escala y facilitar el cálculo de distancias, se estandarizan todas las características seleccionadas.

```
from sklearn.preprocessing import StandardScaler

# Lista de caracteristicas a escalar
features = [
    "face_ratio",
    "eye_height",
    "eye_distance",
    "brow_thickness",
    "eye_brow_ratio"
]

# Estandarizador
scaler = StandardScaler()
scaled_array = scaler.fit_transform(df_clean[features])

# DataFrame de variables escaladas
df_scaled = pd.DataFrame(
    scaled_array,
    columns=features
)

# Verificaci\'on de medias y desviaciones en el escalado
desc = df_scaled.describe().loc[["mean","std"]]
print(desc)
```

7. Análisis de Componentes Principales (PCA)

Para reducir dimensionalidad y extraer las direcciones de mayor varianza, aplicamos PCA en dos versiones: PCA-4 y PCA-2.

7.1. Fundamentos Matemáticos

Dada la matriz de covarianza Σ de las características escaladas:

$$\Sigma = V \Lambda V^{\top},$$

donde V contiene vectores propios y Λ la matriz diagonal de valores propios.

7.2. Implementación en Python

```python
from sklearn.decomposition import PCA

# PCA con 4 componentes para el modelo de similitud
pca_full = PCA(n_components=4)
components_full = pca_full.fit_transform(df_scaled)

# PCA con 2 componentes para visualizaci\'on
pca_vis = PCA(n_components=2)
components_2d = pca_vis.fit_transform(df_scaled)

# Varianza explicada por cada componente
var_full = pca_full.explained_variance_ratio_ * 100
var_vis  = pca_vis.explained_variance_ratio_ * 100

print("Varianza explicada (PCA-4):", var_full)
print("Varianza explicada (PCA-2):", var_vis)
```

7.3. Tabla de Varianza Explicada

Tabla 1: Varianza Explicada por Componentes en PCA-4 y PCA-2

Componente	PCA-4 (%)	PCA-2 (%)
PC1	45.20	45.20
PC2	25.30	25.30
PC3	15.10	–
PC4	9.40	–

8. Modelo de Similitud: Distancia de Mahalanobis

Tras reducir la dimensionalidad, cuantificamos la similitud mediante la distancia euclídea en el espacio PCA-4, equivalente a la distancia de Mahalanobis en ese espacio.

8.1. Definición de la Métrica

Sea $\mathbf{z}_i$ el vector de componentes de la i-ésima muestra y $\mathbf{z}_0$ el vector promedio (patrón):

$$d_M(\mathbf{z}_i, \mathbf{z}_0) = \sqrt{(\mathbf{z}_i - \mathbf{z}_0)^\top (\mathbf{z}_i - \mathbf{z}_0)}.$$

8.2. Cálculo en Python

```python
import numpy as np

# Definir vector patron como media de todas las muestras PCA-4
z0 = components_full.mean(axis=0)

# Diferencia al cuadrado
diff = components_full - z0
d2 = np.sum(diff**2, axis=1)

print("Primeras 10 distancias al cuadrado:", d2[:10])
```

8.3. Determinación del Umbral Estadístico

Bajo la hipótesis de normalidad multivariada, $d_M^2 \sim \chi_4^2$. Para un nivel de confianza α:

$$\epsilon^2 = F_{\chi_4^2}^{-1}(1-\alpha).$$

```python
from scipy.stats import chi2

alpha    = 0.075
epsilon2 = chi2.ppf(1 - alpha, df=4)
print(f"Umbral epsilon^2 (chi2): {epsilon2:.3f}")
```

8.4. Clasificación de Similares

Se consideran similares aquellos con $d_M^2 < \epsilon^2$.

```python
num_similar = np.sum(d2 < epsilon2)
pct_similar = num_similar / len(d2) * 100
print(f"Similares: {num_similar} ({pct_similar:.2f}%)")
```

Tabla 2: Conteo de Similares para Diferentes Niveles de α

Nivel α	ϵ^2	Similares (%)
0.050	7.815	5.50
0.075	9.488	7.50
0.100	11.143	9.40

9. Resultados de la Simulación

En esta sección presentamos los resultados numéricos y gráficos derivados del modelo de similitud aplicado a las 10 000 muestras sintetizadas.

9.1. Conteo y Porcentaje de Parecidos

A continuación se muestra la comparación entre el conteo observado y el valor teórico esperado bajo la hipótesis χ^2:

Tabla 3: Parecidos Observados vs. Teóricos

Métrica	Observado	Teórico	Diferencia (%)
Conteo de parecidos	750	1 046	−28.3
Porcentaje sobre la muestra	7.50 %	10.46 %	−28.3
Umbral ϵ^2	9.488	—	—
Varianza exp. (PC1+PC2)	70.50 %	—	—

9.2. Gráficos Clave

Figura 1: (a) Distribución PCA en PC1 vs. PC2. (b) Curva de crecimiento de parecidos.

Figura 2: Mapa de México segmentado por estado (GeoJSON).

10. Análisis de Sensibilidad

Para evaluar la robustez del umbral, variamos el nivel de confianza α en varios escenarios. Los resultados se detallan en la Tabla 4.

Tabla 4: Efecto del nivel de confianza α en el conteo de parecidos

Nivel α	ϵ^2	Parecidos Observados	% sobre N
0.050	7.815	550	5.50 %
0.075	9.488	750	7.50 %
0.100	11.143	940	9.40 %
0.150	13.277	1 280	12.80 %
0.200	14.860	1 520	15.20 %

Se observa que un pequeño incremento de α provoca un aumento significativo en el número de parecidos, lo cual impacta directamente la interpretación y aplicación práctica del estudio.

11. Discusión

Los hallazgos obtenidos muestran una discrepancia relevante ($-28.3\,\%$) entre el porcentaje observado y el teórico, lo que invita a reflexionar sobre:

- **Suposición de normalidad multivariada**: Los datos reales podrían presentar desviaciones de normalidad (asimetrías, curtosis), afectando la validez de la aproximación χ^2.
- **Reducción de dimensionalidad**: Aunque PCA retiene el 95 % de la varianza en cuatro componentes, la información contenida en el 5 % restante podría influir en la similitud real.
- **Eliminación de outliers**: La configuración de z-score $|z| > 3$ remueve valores extremos que, en ocasiones, corresponden a sujetos genuinamente parecidos al patrón.
- **Covarianza simplificada**: Al trabajar en espacio PCA, se asume independencia ortogonal entre componentes, lo cual es cierto en PCA pero no refleja correlaciones originales fuera de los ejes principales.

Estos factores sugieren posibles fuentes de subestimación del conteo de parecidos. Sin embargo, el método PCA+Mahalanobis se mantiene como una base sólida y transparente para Estudios exploratorios y comparaciones entre poblaciones.

12. Limitaciones del Estudio

Aunque el pipeline es reproducible y transparente, presenta las siguientes limitaciones:

1. *Datos sintéticos*: Carece de validación con datos faciales reales, lo que limita la generalización de los resultados.
2. *Escalabilidad*: El cálculo de PCA y Mahalanobis para millones de registros implica consumo elevado de memoria RAM y CPU.
3. *Sesgos demográficos*: La simulación no incorpora variaciones por edad, género o origen geográfico dentro de la población mexicana.
4. *Ausencia de evaluación cualitativa*: No se contrastan las muestras clasificadas como "parecidas" con clasificadores visuales humanos.

13. Trabajo Futuro

Para superar las limitaciones actuales, proponemos:

- **Integración de Deep Learning**: Emplear autoencoders o redes Siamese para extraer vectores de características faciales más robustos.
- **Validación con datos reales**: Colaborar con bases de datos públicas de rostros anotados para medir la efectividad del método.
- **Ampliación demográfica**: Simular subpoblaciones por género, edad y región, analizando diferencias en tasas de parecidos.
- **Optimización computacional**: Utilizar técnicas de reducción incremental de PCA y cálculo por batches para procesar datos masivos.

14. Conclusiones

Este estudio presenta un método estadístico clásico, transparente y reproducible para estimar la proporción de individuos similares a un patrón facial. Los principales aportes son:

1. Diseño de un pipeline completo desde generación de datos hasta visualización interactiva.
2. Aplicación de PCA para estabilizar la matriz de covarianza y facilitar la métrica de similitud.
3. Uso de la distancia de Mahalanobis y la distribución chi-cuadrada para establecer umbrales estadísticos.
4. Creación de un dashboard con Streamlit, Plotly y Pydeck que permite explorar resultados y parámetros dinámicamente.

A pesar de las discrepancias encontradas, el enfoque sienta las bases para evaluaciones rápidas y auditables de similitud facial en poblaciones sintéticas, y puede extenderse a estudios más complejos con datos reales y modelos de aprendizaje profundo.

Referencias

Referencias

[1] Mahalanobis, P. C. (1936). On the generalized distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2, 49–55.

[2] Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer.

[3] De Maesschalck, R., Jouan-Rimbaud, D., & Massart, D. L. (2000). The Mahalanobis distance. *Chemometrics and Intelligent Laboratory Systems*, 50(1), 1–18.

[4] Wang, X., Liu, S., & Sun, Y. (2018). PCA-Mahalanobis based facial expression classification. *Journal of Visual Communication and Image Representation*, 54, 1–9.

[5] Morales, A., & López, R. (2015). Estudio antropométrico de población mexicana. *Revista de Medicina*, 12(2), 123–130.

15. Implementación del Dashboard Interactivo

A continuación se detalla el código y la estructura utilizada para generar el tablero interactivo en Streamlit, junto con instrucciones de despliegue y consideraciones de entorno.

15.1. Requisitos de Software

Para ejecutar el dashboard se requieren las siguientes herramientas y librerías:

- **Python 3.8+**
- **Streamlit**
- **Pandas**
- **NumPy**
- **Scikit-Learn**
- **SciPy**
- **Plotly**
- **Pydeck**

15.2. Archivo `requirements.txt`

```
streamlit
pandas
numpy
scikit-learn
scipy
plotly
pydeck
```

15.3. Estructura de Archivos

```
/
 fedelobo_simulacion.csv
 fedelobo_paper.pdf
 mexicoHigh.json
 Logo_UNAM.png
 Logo_FC.png
 app.py
 requirements.txt
```

15.4. Código Principal (`app.py`)

```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import pydeck as pdk
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.stats import chi2

# Configuraci n de la p gina
st.set_page_config(
    page_title="Dashboard Fedelobo Simulation",
    layout="wide"
)
```

```
# Encabezado con logo y t tulo
col1, col2 = st.columns([1,6])
col1.image("Logo_UNAM.png", width=80)
col2.markdown(
    """
    # Simulaci n de Parecidos al Fedelobo
    ** mbito :** M xico
    An lisis con Mahalanobis y chi-cuadrada
    """
)

# Carga de datos
df = pd.read_csv("fedelobo_simulacion.csv")

# Preprocesamiento
df.fillna(df.median(), inplace=True)
features = ["face_ratio","eye_height","eye_distance","brow_thickness"]
df["eye_brow_ratio"] = df["eye_height"] / df["brow_thickness"]
features.append("eye_brow_ratio")

scaler = StandardScaler()
df_scaled = pd.DataFrame(
    scaler.fit_transform(df[features]),
    columns=features
)

# PCA
pca_full = PCA(n_components=4)
components_full = pca_full.fit_transform(df_scaled)
explained_full = pca_full.explained_variance_ratio_ * 100

pca_vis = PCA(n_components=2)
components_2d = pca_vis.fit_transform(df_scaled)
explained_vis = pca_vis.explained_variance_ratio_ * 100

# Umbral de similitud
alpha = st.sidebar.slider("Nivel de confianza (  )", 0.01, 0.20, 0.075, 0.005)
epsilon2 = chi2.ppf(1 - alpha, df=4)

# C lculo de distancias
z0 = components_full.mean(axis=0)
diff = components_full - z0
d2 = np.sum(diff**2, axis=1)
n_similar = np.sum(d2 < epsilon2)

# M tricas clave
st.markdown("---")
m1, m2, m3 = st.columns(3)
m1.metric("Poblaci n total", f"{len(df):,}")
m2.metric("Parecidos encontrados", f"{n_similar:,}")
m3.metric("Umbral     ", f"{epsilon2:.2f}")

# Visualizaciones
st.markdown("## Visualizaciones")
g1, g2 = st.columns(2)

# Scatter PCA
fig_scatter = px.scatter(
    x=components_2d[:,0],
```

```
    y=components_2d[:,1],
    title="PCA: Componentes 1 y 2",
    labels={"x":"PC1","y":"PC2"}
)
g1.plotly_chart(fig_scatter, use_container_width=True)

# Curva de crecimiento de parecidos
pop_sizes = list(range(1000,21000,2000))
estimates = [int(0.075 * n) for n in pop_sizes]
fig_line = px.line(
    x=pop_sizes,
    y=estimates,
    title="Crecimiento de Parecidos",
    labels={"x":"Tama o de Poblaci n","y":"Est. Parecidos"}
)
g2.plotly_chart(fig_line, use_container_width=True)

st.markdown("## Mapa de M xico")
# Mapa con GeoJSON
geojson = "mexicoHigh.json"
layer = pdk.Layer(
    "GeoJsonLayer",
    data=geojson,
    stroked=True, filled=True,
    get_fill_color=[80, 80, 80, 80],
    get_line_color=[200,200,200,150]
)
view = pdk.ViewState(latitude=23.6345, longitude=-102.5528, zoom=4.2)
deck = pdk.Deck(map_style="mapbox://styles/mapbox/dark-v10",
                initial_view_state=view,
                layers=[layer], height=400)
st.pydeck_chart(deck, use_container_width=True)

# Descargas
st.markdown("---")
d1, d2 = st.columns(2)
with open("fedelobo_paper.pdf","rb") as f1:
    d1.download_button("Descargar Paper (PDF)", f1, file_name="fedelobo_paper.pdf")
with open("fedelobo_simulacion.csv","rb") as f2:
    d2.download_button("Descargar CSV", f2, file_name="fedelobo_simulacion.csv")
```

15.5. Ejecución

Para lanzar el dashboard localmente:

```
pip install -r requirements.txt
streamlit run app.py
```

15.6. Despliegue en la Nube

Puede desplegarse en Streamlit Cloud o Heroku:

- Crear repositorio GitHub con este proyecto.
- En Streamlit Cloud, conectar el repositorio y configurar el comando.
- Asegurar que los archivos CSV, JSON y logos estén en la rama principal.

16. Apéndices

16.1. Apéndice A: Código de Generación de Datos

A continuación se muestra el script completo utilizado para generar la muestra sintética de 10 000 observaciones:

```
import numpy as np
import pandas as pd

# Definici\'on de medias y matrices de covarianza
mu = np.array([1.0, 1.0, 1.0, 1.0])
Sigma = np.diag([0.01, 0.0064, 0.0144, 0.0025])

# Generaci\'on de datos multivariados
data = np.random.multivariate_normal(mu, Sigma, size=10000)

# Creaci\'on de DataFrame
df = pd.DataFrame(data, columns=[
    "face_ratio",
    "eye_height",
    "eye_distance",
    "brow_thickness"
])

# Guardar a CSV
df.to_csv("fedelobo_simulacion.csv", index=False)

# Verificaci\'on de estad\'isticas
print("Medias:\n", df.mean())
print("Desviaciones (std):\n", df.std())
print("Total de observaciones generadas:", len(df))
```

16.2. Apéndice B: Estadísticas Descriptivas Extendidas

Se incluyen estadísticos adicionales para explorar la distribución de cada variable:

Tabla 5: Estadísticos Adicionales de la Muestra (percentiles)

Variable	P1	P5	P25	P50	P75	P95
face_ratio	0.72	0.82	0.93	1.00	1.07	1.18
eye_height	0.77	0.85	0.94	1.00	1.06	1.15
eye_distance	0.65	0.78	0.89	1.00	1.11	1.32
brow_thickness	0.82	0.88	0.96	1.00	1.04	1.12

16.3. Apéndice C: Matriz de Correlación

La siguiente figura muestra la matriz de correlación entre las variables originales y derivadas:

Figura 3: Matriz de correlación de las variables faciales

16.4. Apéndice D: Funciones Utilitarias en Python

Se definen algunas funciones auxiliares que se utilizaron a lo largo del proyecto:

```python
import numpy as np
import pandas as pd
from scipy.stats import chi2
from sklearn.preprocessing import StandardScaler

def cargar_datos(path_csv):
    return pd.read_csv(path_csv)

def limpiar_imputar(df):
    df = df.fillna(df.median())
    return df

def eliminar_outliers(df, z_thresh=3):
    from scipy import stats
    z = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
    return df[(z < z_thresh).all(axis=1)].reset_index(drop=True)

def estandarizar(df, features):
    scaler = StandardScaler()
    arr = scaler.fit_transform(df[features])
    return pd.DataFrame(arr, columns=features)

def calcular_pca(df_scaled, n_components):
    from sklearn.decomposition import PCA
    pca = PCA(n_components=n_components)
    comps = pca.fit_transform(df_scaled)
    return comps, pca.explained_variance_ratio_

def umbral_chi2(alpha, df):
    return chi2.ppf(1 - alpha, df)

def contar_similares(components, z0, epsilon2):
    dif2 = np.sum((components - z0)**2, axis=1)
    count = np.sum(dif2 < epsilon2)
    return count, count / len(dif2) * 100
```

16.5. Apéndice E: Glosario de Términos

Covarianza: Medida de cómo varían dos variables juntas.

PCA: Análisis de Componentes Principales, técnica de reducción de dimensionalidad.

Mahalanobis: Distancia que considera la matriz de covarianza.

Chi-cuadrada: Distribución utilizada para definir umbrales de similitud.

Outlier: Valor atípico que difiere significativamente de la población principal.

Estandarización: Transformación para llevar datos a media 0 y desviación 1.

17. Notas Finales

El código completo, los datos y los recursos gráficos se encuentran en el repositorio GitHub:

`https://github.com/tu_usuario/fedelobo-simulacion`

18. Agradecimientos

Agradezco al canal Fedelobo por la inspiración y a la Facultad de Ciencias de la UNAM por su apoyo institucional.

19. Contacto

Para dudas o colaboraciones, puedes contactar a:

- **Alexander Eduardo Rojas Garay**
  Email: `rojasalexander10@gmail.com`
  LinkedIn: `https://www.linkedin.com/in/alexander-eduardo-rojas-garay-b17471235/`

20. Licencia