****

**Título de la obra:** Manual técnico y flujo de procesamiento CHIRPVIEW (V7) — Línea L35.  
**Autor:** Marco Antonio Viveros Velásquez.  
**Año:** 2025.  
**Naturaleza de la obra:** manual técnico con software asociado.  
**Nombre del desarrollo/software:** CHIRPVIEW (V7).

**Resumen breve:** Este manual documenta el flujo completo para transformar datos CHIRP (archivo SEG/SEGY) y registros de navegación (CSV/ODC) en productos reproducibles para análisis geocientífico (perfil distancia–profundidad, mapa interactivo, y salidas vectoriales KML/KMZ/SHP/GeoJSON), incluyendo controles de calidad, hashes de integridad y empaquetado listo para depósito en INAPI. La versión V7 anota coordenadas UTM 18S (EPSG:32718) de inicio y término en el perfil, genera un archivo ODC derivado y crea un índice HTML de resultados.

**1. Resumen ejecutivo**

Esta guía define un procedimiento robusto y verificable para procesar la **Línea CHIRP L35** del Cañón Submarino de San Antonio (SAC). El flujo se implementa en **Python** y prioriza: (i) reproducibilidad (entorno, dependencias, rutas determinísticas), (ii) trazabilidad (CSV/ODC fuente, ODC derivado, mapa interactivo y vectoriales), y (iii) verificabilidad (checksums SHA‑256 y archivo índice). El **script base es L35masterV7.py** y se invoca desde un **BAT** que estandariza la ejecución. Como innovación práctica, V7 anota en el gráfico final las coordenadas **UTM 18S** de los extremos de la línea con tipografía legible y alineada a márgenes, además de producir **KML/KMZ/SHP/GeoJSON/JSON** y un **ODC** nuevo cuando faltan o se desean homogeneizar las posiciones. Este manual se inscribe en INAPI como obra literaria técnica; el código se deposita como software adjunto (archivo comprimido y listado de dependencias), lo que protege tanto el texto como la implementación. La obra se acompaña de un conjunto de productos de ejemplo que demuestran la ejecución efectiva.

**2. Introducción**

Este manual documenta el desarrollo y la operación de **CHIRPVIEW (V7)** aplicado a la **Línea L35**, un flujo técnico diseñado para transformar datos sísmicos CHIRP crudos en productos reproducibles y auditables para análisis geocientífico y registro ante **INAPI**. El documento integra, en una sola obra, la metodología, el guion de procesamiento y las pautas de control de calidad necesarias para asegurar **reproducibilidad**, **trazabilidad** e **integridad** de resultados. CHIRPVIEW (V7) ingiere archivos **SEG/SEGY** y navegación **CSV/ODC**, valida rangos, proyecta a **UTM 18S (EPSG:32718)**, calcula distancia acumulada y profundidad (TWT→Z con V=1500 m/s), y genera un perfil distancia–profundidad con anotación de coordenadas UTM en los extremos. Además, produce un mapa interactivo **Folium**, exportes vectoriales **KML/KMZ/SHP/GeoJSON**, tablas de QA y un índice HTML para inspección rápida, junto con **checksums** para verificación. La obra propone una estructura de carpetas clara, dependencias explícitas y un **BAT** de ejecución para usuarios no técnicos, favoreciendo el despliegue en entornos Windows y la replicación controlada. Finalmente, el manual ofrece lineamientos de licenciamiento, empaquetado de anexos y buenas prácticas para envío a INAPI, así como una hoja de ruta de extensiones futuras (p. ej., GeoTIFF del perfil o empaquetado como biblioteca). Con ello, se garantiza una base sólida, verificable y escalable para el procesamiento CHIRP en campañas presentes y futuras.

**3. Objetivo**

Establecer y documentar un flujo **estándar, claro y repetible** para convertir datos CHIRP crudos (SEG/SEGY) y navegación (CSV/ODC) en productos listos para análisis, visualización y archivo. El manual y el software **CHIRPVIEW (V7)** buscan ser **auto‑contenibles**: cualquier técnico podrá reproducir resultados siguiendo la estructura de carpetas, instalando dependencias y ejecutando el BAT. En términos de propiedad intelectual, la obra garantiza que la metodología, decisiones de diseño (parámetros, proyecciones, filtros) y la implementación en V7 queden descritas con precisión suficiente para su **registro en INAPI**, sin necesidad de incluir el código dentro del cuerpo del manual.

**4. Alcance y supuestos**

El flujo aplica a perfiles **CHIRP marinos** con navegación georreferenciada. Entradas esperadas:

* **SEG/SEGY** con trazas y cabeceras legibles por ObsPy.
* **CSV/ODC** con lat/lon (WGS84). Si no existen, V7 sintetiza una navegación realista para pruebas.  
  Supuestos: área litoral central de Chile; latitudes y longitudes dentro de rangos válidos; proyección de trabajo **UTM 18S (EPSG:32718)**; velocidad del sonido **VEL\_MS = 1500 m/s**; estiramiento vertical **Z\_MULT = 10**. Resultados: perfil distancia–profundidad, mapa Folium, vectoriales **KML/KMZ/SHP/GeoJSON**, **JSON** de puntos, **ODC** derivado, CSV de QA y archivo **index.html** con accesos rápidos. No se incluyen rutinas de interpretación geológica; el flujo es **pre‑analítico** y de estandarización. El alcance contempla ejecución local en Windows 10/11, con o sin entorno virtual.

**5. Estructura de carpetas**

Raíz del proyecto: C:\\Python2025\_SAC\\GITHUBL58

GITHUBL58\

├─ .venv\ (entorno virtual, opcional)

├─ Handbook\ (manuales, README)

├─ LinesDat.L35\ (entradas: SEG/ODC/CSV)

│ ├─ 20190817112135.seg (obligatorio)

│ ├─ 20190817112135.odc (opcional)

│ └─ 20190817112135.csv (opcional)

├─ out\L35\ (salidas automáticas)

├─ src\

│ └─ L35masterV7.py (script principal)

└─ run\_L35masterV7.bat (lanzador)

Esta organización asegura trazabilidad (insumos vs salidas), facilita empaquetado para registro y reduce errores por rutas.

**6. Requisitos y dependencias**

* **Python 3.10+** (Windows).
* Bibliotecas: **obspy**, **numpy**, **pandas**, **matplotlib**, **pyproj**, **folium**.
* Opcionales para Shapefile/GeoJSON avanzados: **geopandas**, **shapely**, **pyogrio**.  
  Se recomienda un entorno virtual .venv y el archivo BAT provisto. La ausencia de GeoPandas/Shapely no impide el flujo: V7 genera **KML/KMZ** y un **GeoJSON mínimo** sin dependencias.

**7. Instalación y puesta en marcha**

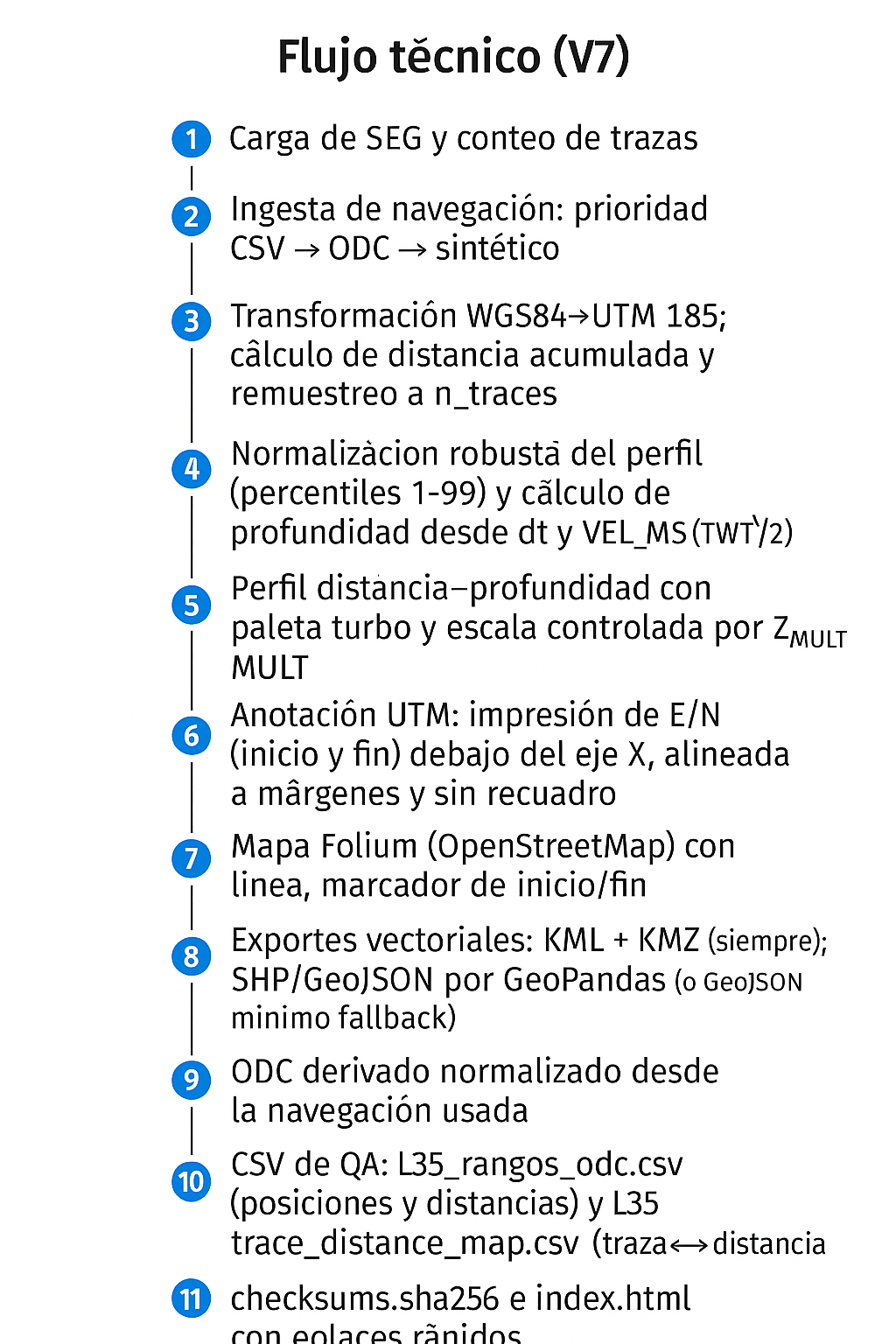
1. Crear/activar entorno (opcional): python -m venv .venv y ..\\..\\.venv\\Scripts\\activate.
2. Instalar dependencias: pip install obspy numpy pandas matplotlib pyproj folium y, si desea SHP/GeoJSON nativo, pip install geopandas shapely pyogrio.
3. Verificar entradas en LinesDat.L35.
4. Ejecutar con doble clic run\_L35masterV7.bat **o** desde PowerShell:  
   cd C:\\Python2025\_SAC\\GITHUBL58 → .\\run\_L35masterV7.bat.  
   El BAT registra stdout/stderr en out\\L35\\run\_L35masterV7.log y abre el índice HTML al terminar.

**8. Datos de entrada y validaciones**

* **SEG/SEGY:** formato estándar; ObsPy extrae n\_traces, delta y datos.
* **CSV/ODC:** columnas autodetectadas para lat/lon y opcionalmente date/time. Se filtran rangos plausibles (Chile central) y se limpian NaN.
* **Navegación sintética:** si no hay CSV/ODC válidos, V7 genera posiciones (WGS84) centradas en San Antonio, con timestamps regulares para pruebas controladas.
* **Proyección:** conversión a **UTM 18S (EPSG:32718)** vía **pyproj**.
* **Remuestreo:** la distancia acumulada se interpola al número de trazas para alinear X (metros) con el eje de trazas del SEG.

**9. Flujo técnico (V7)**

1. Carga de SEG y conteo de trazas.
2. Ingesta de navegación: prioridad **CSV → ODC → sintético**.
3. Transformación WGS84→UTM 18S; cálculo de distancia acumulada y remuestreo a n\_traces.
4. Normalización robusta del perfil (percentiles 1–99) y cálculo de profundidad desde dt y **VEL\_MS** (TWT/2).
5. **Perfil distancia–profundidad** con paleta **turbo** y escala controlada por **Z\_MULT**.
6. **Anotación UTM**: impresión de **E/N** (inicio y fin) **debajo** del eje X, alineada a márgenes y sin recuadro.
7. **Mapa Folium** (OpenStreetMap) con línea, marcador de inicio/fin.
8. **Exportes vectoriales**: KML + KMZ (siempre); SHP/GeoJSON por GeoPandas (o GeoJSON mínimo fallback).
9. **ODC derivado** normalizado desde la navegación usada.
10. **CSV de QA**: L35\_rangos\_odc.csv (posiciones y distancias) y L35\_trace\_distance\_map.csv (traza↔distancia).
11. **checksums.sha256** e **index.html** con enlaces rápidos.

****

**10. Productos y formatos**

* **Perfil**: L35\_profile\_master.jpg/.svg. Ejes en metros; colormap turbo; Z\_MULT predeterminado 10; UTM de extremos bajo el eje X.
* **Mapa interactivo**: L35\_track\_master.html (Folium).
* **Vectoriales**: L35\_track.kml, L35\_track.kmz, L35\_track.shp (si hay GeoPandas) y L35\_track.geojson.
* **JSON**: L35\_track.json (puntos + metadatos básicos).
* **Navegación derivada**: 20190817112135\_v7.odc.
* **QA**: L35\_rangos\_odc.csv, L35\_trace\_distance\_map.csv.
* **Integridad**: checksums.sha256.
* **Índice**: index.html para recorrer resultados.

**11. Detalles cartográficos y parámetros**

* **CRS de trabajo:** UTM 18S (**EPSG:32718**).
* **Etiquetas UTM**: texto negro de alto contraste, alineado a los márgenes del panel, fuera del área de datos.
* **Profundidad**: profundidad\_max = n\_samples \* dt \* VEL\_MS / 2.
* **Distancia**: acumulada tras filtrar outliers de paso por umbral robusto; rejilla de X adaptativa mediante \_choose\_step para legibilidad.
* **Colores**: CMAP = "turbo"; puede adaptarse según preferencias de publicación.

**12. Control de calidad (QA)**

* Validar que la **distancia** crece monótonamente y que el mapa coincida con la zona esperada.
* Revisar run\_L35masterV7.log ante comportamientos anómalos.
* Confirmar que **checksums** listan todos los productos entregables.
* Si SHP no se escribe, instalar dependencias geo; de lo contrario, usar GeoJSON/KMZ.
* Con CSV/ODC propios, deshabilitar navegación sintética (mantener archivos presentes en LinesDat.L35).

**13. Ejecución con BAT y logging**

El archivo run\_L35masterV7.bat intenta usar ..\\.venv\\Scripts\\python.exe y, si no existe, recurre a python del sistema. Redirige la salida a out\\L35\\run\_L35masterV7.log, reporta **ERRORLEVEL** y abre automáticamente **index.html**, el perfil y el mapa si existen. Esto estandariza la operación para usuarios no técnicos y crea una **pista de auditoría** conveniente para el expediente INAPI.

**14. Empaquetado para registro en INAPI**

Para el depósito, preparar un **ZIP** con:

* **Manual PDF** (exporte de este documento).
* **Código fuente** como software: src\\L35masterV7.py, archivo BAT y requirements.txt (o lista de paquetes).
* **Datos mínimos de ejemplo**: un SEG recortado y CSV/ODC reducidos (si la licencia lo permite) **o** evidencias (imágenes del perfil y mapa).
* **Salidas** en out\\L35 (al menos JPG/SVG del perfil, KML/KMZ/GeoJSON, JSON, ODC derivado, CSV de QA, checksums y index.html).
* **Licencia** sugerida para repositorios (MIT/BSD‑3) y “todos los derechos reservados” para el registro INAPI.  
  Esta estrategia protege la obra escrita y el software, sin exponer datos sensibles.

**15. Autoría, derechos y licenciamiento**

Autor: Marco Antonio Viveros Velásquez.

Para difusión académica adicional (p. ej., GitHub/Zenodo), se sugiere licencia MIT o BSD 3 en el repositorio, manteniendo todos los derechos reservados en el registro INAPI. Las bibliotecas de terceros conservan sus respectivas licencias; se citan en requirements/README.

**16. Declaración de originalidad y uso de terceros**

El flujo V7, diseño del BAT, estructura de salidas y documentación son originales del autor. Se emplean bibliotecas abiertas (ObsPy, NumPy, etc.) conforme a sus licencias. Cualquier dato de terceros usado como ejemplo debe estar autorizado o anonimizado.

**17. Mantenimiento y control de versiones**

**V7** es la **versión base**. Cambios futuros deberán registrarse en un **CHANGELOG**, manteniendo compatibilidad de rutas y formatos. Recomendación: etiquetar releases (V8, V9) y adjuntar diffs de parámetros (CRS, colormap, Z\_MULT, etc.).

**18. Glosario breve**

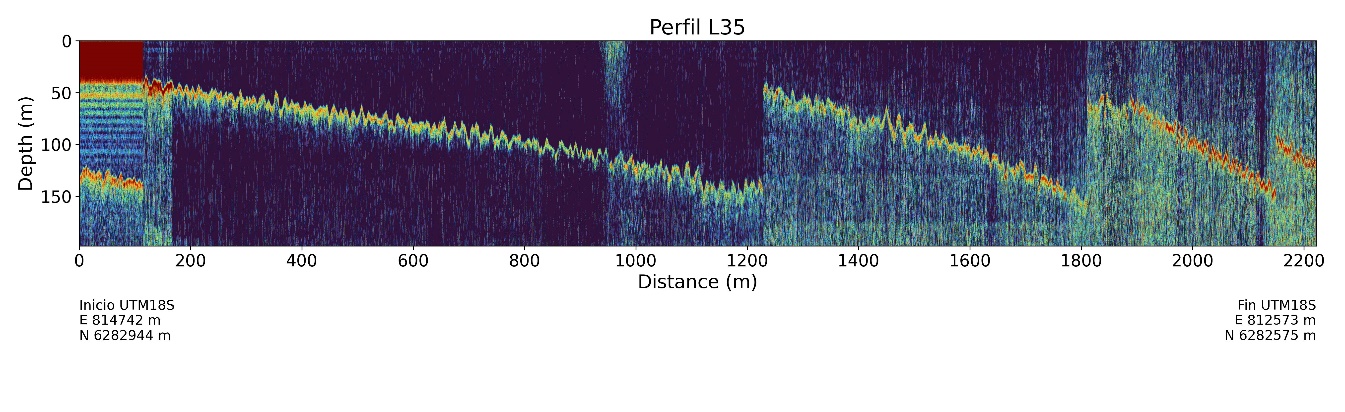
* **CHIRP:** sísmica de alta resolución para perfiles someros.
* **SEG/SEGY:** formato estándar de datos sísmicos.
* **ODC/CSV:** formatos de navegación (fecha/hora/lat/lon).
* **UTM 18S / EPSG:32718:** proyección métrica usada en V7.
* **Folium:** librería para mapas interactivos HTML.
* **Checksums SHA‑256:** huellas para verificar integridad de archivos.

**19. Anexos (sugeridos)**

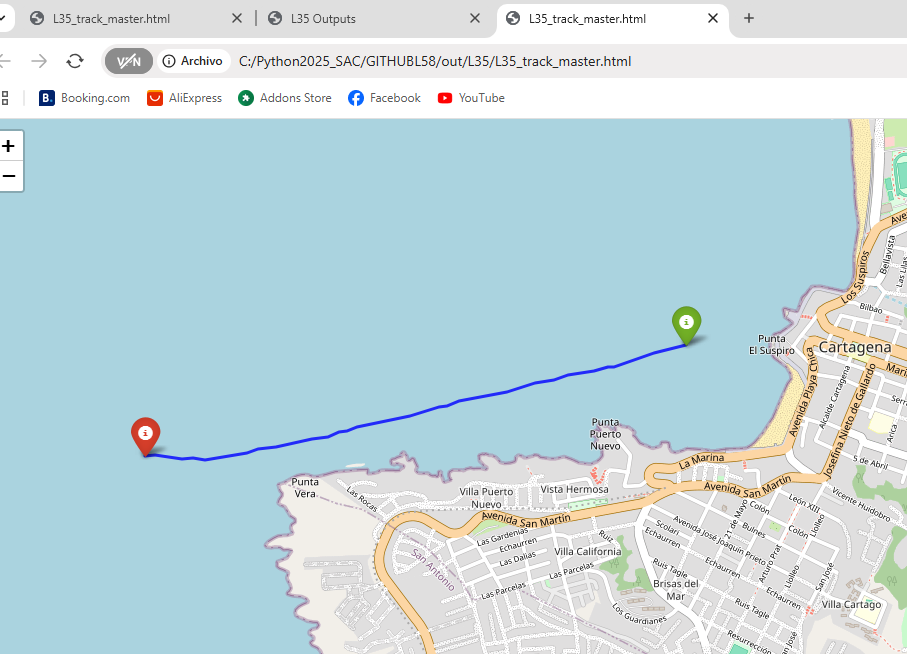
* Capturas del perfil
* Mapas generados por V7.
* Código CHIRPviewV7
* README — Ejecución rápida L35 (V7)

**Nota:** El código fuente **no** se incluye en el cuerpo del manual; se entrega como archivo adjunto para su depósito como software asociado.

### PERFIL L35



### MAPA L35



### CÓDIGO

# -\*- coding: utf-8 -\*-

# L35master.py (con export a Shapefile y KMZ)

# IN :

#   C:\Python2025\_SAC\GITHUBL58\LinesDat.L35\20190817112135.(seg|odc|CSV)

# OUT:

#   C:\Python2025\_SAC\GITHUBL58\out\L35\

#

# Flujo:

# 1) Lee posiciones desde CSV (si existe) o desde ODC.

# 2) lat/lon -> metros (UTM 18S, EPSG:32718), filtra outliers, distancia acumulada.

# 3) Remuestrea distancia a Nº de trazas del SEG y genera perfil (Distance & Depth).

# 4) Genera mapa Folium.

# 5) Exporta línea de navegación a Shapefile, KMZ, GeoJSON y JSON.

# [V7] Igual que V6 + etiquetas UTM (E/N) sin caja, en negro, más grandes y

#      alineadas con los márgenes izquierdo y derecho, debajo del eje X.

import os, sys, subprocess, zipfile, json, hashlib

import numpy as np

import pandas as pd

import folium

import matplotlib.pyplot as plt

from matplotlib.ticker import MultipleLocator, ScalarFormatter

from obspy import read

from pyproj import Transformer

from datetime import datetime, timedelta

# ---- INTENTAR importar geopandas/shapely para SHP (opcionalmente) ----

try:

    import geopandas as gpd

    from shapely.geometry import LineString

    \_GEO\_OK = True

except Exception:

    \_GEO\_OK = False

# ---------------- Rutas ----------------

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))                    # ...\GITHUBL58\src

ROOT\_DIR = os.path.abspath(os.path.join(BASE\_DIR, ".."))                 # ...\GITHUBL58

ARCH\_DIR = os.path.join(ROOT\_DIR, "LinesDat.L35")                         # ...\GITHUBL58\LinesDat.L35

OUT\_DIR  = os.path.join(ROOT\_DIR, "out", "L35")                           # ...\GITHUBL58\out\L35

os.makedirs(OUT\_DIR, exist\_ok=True)

SEG\_PATH = os.path.join(ARCH\_DIR, "20190817112135.seg")

ODC\_PATH = os.path.join(ARCH\_DIR, "20190817112135.odc")

CSV\_PATH = os.path.join(ARCH\_DIR, "20190817112135.CSV")                  # opcional

# ---------------- Parámetros ----------------

VEL\_MS  = 1500.0     # velocidad del sonido (m/s) -> para convertir TWT a profundidad

Z\_MULT  = 10.0       # factor vertical para profundidades (1.0 = sin estirar)

CMAP    = "turbo"    # paleta estilo quicklook

DPI     = 300

FIGSIZE = (18, 5.5)

# Fallback para navegación sintética (San Antonio, Chile aprox.)

FALLBACK\_CENTER\_LAT = -33.6000

FALLBACK\_CENTER\_LON = -71.6200

plt.rcParams.update({

    "font.size": 16, "axes.titlesize": 20, "axes.labelsize": 18,

    "xtick.labelsize": 16, "ytick.labelsize": 16

})

def \_open\_file(path: str):

    try:

        if os.name == "nt": os.startfile(path)

        elif sys.platform == "darwin": subprocess.Popen(["open", path])

        else: subprocess.Popen(["xdg-open", path])

    except Exception:

        pass

def \_choose\_step(total\_m: float) -> float:

    for step in (200, 250, 500, 1000, 2000, 5000, 10000, 20000):

        if total\_m / step <= 14:

            return float(step)

    raw = max(1.0, total\_m / 14.0)

    return float(int(np.ceil(raw / 100.0) \* 100))

# ---------- Lectura posiciones desde CSV (si existe) ----------

def parse\_csv\_positions(csv\_path: str) -> pd.DataFrame:

    if not os.path.exists(csv\_path):

        return pd.DataFrame()

    try:

        df = pd.read\_csv(csv\_path, engine="python", encoding\_errors="ignore")

    except Exception:

        return pd.DataFrame()

    low2orig = {c.lower(): c for c in df.columns}

    def pick(keys):

        for k in keys:

            if k in low2orig:

                return low2orig[k]

        for lk, orig in low2orig.items():

            if any(k in lk for k in keys):

                return orig

        return None

    lat\_col  = pick(["lat", "latitude", "latitud"])

    lon\_col  = pick(["lon", "long", "longitude", "longitud"])

    date\_col = pick(["date", "fecha"])

    time\_col = pick(["time", "hora"])

    if not lat\_col or not lon\_col:

        return pd.DataFrame()

    out = pd.DataFrame({

        "lat": pd.to\_numeric(df[lat\_col], errors="coerce"),

        "lon": pd.to\_numeric(df[lon\_col], errors="coerce"),

    })

    if date\_col: out["date"] = df[date\_col].astype(str)

    if time\_col: out["time"] = df[time\_col].astype(str)

    m = out["lat"].between(-90, 10) & out["lon"].between(-180, -50)

    out = out[m].dropna(subset=["lat", "lon"]).reset\_index(drop=True)

    return out

# ---------- Lectura posiciones desde ODC ----------

def parse\_odc\_positions(odc\_path: str) -> pd.DataFrame:

    rows = []

    if not os.path.exists(odc\_path):

        return pd.DataFrame()

    with open(odc\_path, "r", errors="ignore") as f:

        for line in f:

            s = line.strip()

            if not s: continue

            if ",151," not in s: continue

            parts = [p.strip() for p in s.split(",")]

            try:

                i\_date = next(i for i,p in enumerate(parts) if "/" in p)

                i\_time = i\_date + 1

                lat = float(parts[i\_time + 1]); lon = float(parts[i\_time + 2])

                rows.append({"date": parts[i\_date], "time": parts[i\_time], "lat": lat, "lon": lon})

            except Exception:

                continue

    df = pd.DataFrame(rows)

    if df.empty: return df

    m = df["lat"].between(-90, 10) & df["lon"].between(-180, -50)

    return df[m].reset\_index(drop=True)

# ---------- Sintetizar navegación si no hay CSV/ODC ----------

def synthesize\_navigation(n\_points: int) -> pd.DataFrame:

    if n\_points < 2: n\_points = 2

    dlat = 2e-5

    dlon = 2e-5 \* np.cos(np.deg2rad(FALLBACK\_CENTER\_LAT))

    lat0 = FALLBACK\_CENTER\_LAT - (n\_points//2)\*dlat

    lon0 = FALLBACK\_CENTER\_LON - (n\_points//2)\*dlon

    lat = lat0 + dlat\*np.arange(n\_points)

    lon = lon0 + dlon\*np.arange(n\_points)

    t0 = datetime(2019, 8, 17, 11, 21, 35)

    ts = [t0 + timedelta(seconds=i) for i in range(n\_points)]

    df = pd.DataFrame({

        "date": [dt.strftime("%Y/%m/%d") for dt in ts],

        "time": [dt.strftime("%H:%M:%S") for dt in ts],

        "lat": lat,

        "lon": lon

    })

    print(f"[Aviso] Navegación sintetizada (n={n\_points}) centrada en San Antonio, CL.")

    return df

# ---------- Lat/Lon -> distancia en metros; remuestreo a nº de trazas ----------

def latlon\_to\_distance(df\_pos: pd.DataFrame, n\_traces: int):

    if df\_pos.empty:

        return None, pd.DataFrame()

    tr = Transformer.from\_crs("EPSG:4326", "EPSG:32718", always\_xy=True)  # UTM 18S

    x, y = tr.transform(df\_pos["lon"].values, df\_pos["lat"].values)

    dx = np.diff(x); dy = np.diff(y)

    step = np.hypot(dx, dy)

    med = np.median(step) if len(step) else 0.0

    thr = max(5.0, 10.0 \* med) if med > 0 else 5.0

    step\_clipped = np.clip(step, 0.0, thr)

    dist = np.insert(np.cumsum(step\_clipped), 0, 0.0)

    t\_src = np.linspace(0, 1, len(dist))

    t\_tgt = np.linspace(0, 1, n\_traces)

    dist\_traces = np.interp(t\_tgt, t\_src, dist)

    df\_pos = df\_pos.copy()

    df\_pos["dist\_m"] = dist

    return dist\_traces, df\_pos

# ---------- Perfil desde SEG con X en metros (+ UTM en extremos) ----------

def plot\_profile\_with\_distance(seg\_path: str, dist\_m: np.ndarray, out\_dir: str,

                               utm\_start\_end=None):

    """

    utm\_start\_end: tuple (E\_ini, N\_ini, E\_fin, N\_fin) en EPSG:32718.

    """

    st = read(seg\_path, format="SEGY", unpack\_trace\_headers=True)

    data = np.array([tr.data for tr in st], dtype=float)

    n\_traces, n\_samples = data.shape

    dt = st[0].stats.delta

    depth\_max\_m = n\_samples \* dt \* VEL\_MS / 2.0

    p1, p99 = np.percentile(data, [1, 99])

    p99 = p99 if p99 > p1 else (p1 + 1.0)

    data = np.clip(data, p1, p99)

    data = (data - p1) / (p99 - p1)

    total\_m = float(dist\_m[-1] - dist\_m[0])

    extent = [float(dist\_m[0]), float(dist\_m[-1]), depth\_max\_m \* Z\_MULT, 0.0]

    fig, ax = plt.subplots(figsize=FIGSIZE)

    ax.imshow(data.T, aspect="auto", cmap=CMAP, extent=extent)

    ax.set\_xlabel("Distance (m)")

    ax.set\_ylabel("Depth (m)")

    ax.set\_title("Perfil L35")

    step = \_choose\_step(total\_m)

    ax.xaxis.set\_major\_locator(MultipleLocator(step))

    ax.xaxis.set\_major\_formatter(ScalarFormatter(useMathText=False))

    ax.ticklabel\_format(style="plain", axis="x")

    if total\_m / step > 12:

        ax.tick\_params(axis="x", labelrotation=45)

    # --- [V7] Etiquetas UTM DEBAJO del eje X, alineadas a márgenes ---

    if utm\_start\_end is not None:

        e1, n1, e2, n2 = utm\_start\_end

        # Colocar en coordenadas del eje (0 = borde izq., 1 = borde der.)

        x\_left\_axes, x\_right\_axes = 0.0, 1.0

        y\_frac = -0.26  # más negativo = más abajo

        txtL = f"Inicio UTM18S\nE {int(round(e1))} m\nN {int(round(n1))} m"

        txtR = f"Fin UTM18S\nE {int(round(e2))} m\nN {int(round(n2))} m"

        ax.text(x\_left\_axes,  y\_frac, txtL, transform=ax.transAxes,

                ha="left", va="top", color="black", fontsize=13)

        ax.text(x\_right\_axes, y\_frac, txtR, transform=ax.transAxes,

                ha="right", va="top", color="black", fontsize=13)

    # margen inferior extra para no recortar las etiquetas

    fig.subplots\_adjust(bottom=0.30)

    plt.tight\_layout(rect=(0, 0.10, 1, 1))

    jpg = os.path.join(out\_dir, "L35\_profile\_master.jpg")

    svg = os.path.join(out\_dir, "L35\_profile\_master.svg")

    plt.savefig(jpg, dpi=DPI)

    plt.savefig(svg)

    plt.close()

    print(f"✅ Perfil guardado:\n  {jpg}\n  {svg}")

    \_open\_file(jpg)

    qa\_csv = os.path.join(out\_dir, "L35\_trace\_distance\_map.csv")

    pd.DataFrame({"trace": np.arange(n\_traces), "dist\_m": dist\_m}).to\_csv(qa\_csv, index=False)

    print(f"📝 QA guardado: {qa\_csv}")

# ---------- Export vectorial: Shapefile + KMZ + GeoJSON + JSON ----------

def export\_track\_products(df\_pos: pd.DataFrame, out\_dir: str, name\_prefix: str = "L35\_track"):

    dfp = df\_pos.dropna(subset=["lat", "lon"]).copy()

    if len(dfp) < 2:

        print("[Aviso] Pocos puntos para exportar línea.")

        return

    # --- KML + KMZ ---

    kml\_path = os.path.join(out\_dir, f"{name\_prefix}.kml")

    kmz\_path = os.path.join(out\_dir, f"{name\_prefix}.kmz")

    coords\_txt = " ".join([f"{lon:.8f},{lat:.8f},0" for lat, lon in zip(dfp["lat"], dfp["lon"])])

    kml = f"""<?xml version="1.0" encoding="UTF-8"?>

<kml xmlns="http://www.opengis.net/kml/2.2">

<Document>

  <name>{name\_prefix}</name>

  <Placemark>

    <name>{name\_prefix}</name>

    <Style>

      <LineStyle><color>ff0000ff</color><width>3</width></LineStyle>

    </Style>

    <LineString>

      <tessellate>1</tessellate>

      <coordinates>

        {coords\_txt}

      </coordinates>

    </LineString>

  </Placemark>

</Document>

</kml>"""

    with open(kml\_path, "w", encoding="utf-8") as f:

        f.write(kml)

    with zipfile.ZipFile(kmz\_path, "w", compression=zipfile.ZIP\_DEFLATED) as zf:

        zf.write(kml\_path, arcname=os.path.basename(kml\_path))

    print(f"🗺️  KML guardado: {kml\_path}")

    print(f"🗺️  KMZ guardado: {kmz\_path}")

    # --- Shapefile (si hay geopandas/shapely) ---

    if \_GEO\_OK:

        try:

            line = LineString([(lon, lat) for lat, lon in zip(dfp["lat"], dfp["lon"])])

            gdf = gpd.GeoDataFrame({"name": [name\_prefix]}, geometry=[line], crs="EPSG:4326")

            shp\_path = os.path.join(out\_dir, f"{name\_prefix}.shp")

            gdf.to\_file(shp\_path, driver="ESRI Shapefile")

            print(f"🗂️  Shapefile guardado: {shp\_path}  (se crean .shx, .dbf, .prj)")

            geojson\_path = os.path.join(out\_dir, f"{name\_prefix}.geojson")

            gdf.to\_file(geojson\_path, driver="GeoJSON")

            print(f"🌐  GeoJSON guardado: {geojson\_path}")

        except Exception as e:

            print(f"[Aviso] No se pudo escribir Shapefile/GeoJSON ({e}). ¿Instalaste geopandas/shapely/pyogrio?")

    else:

        print("[Aviso] Geopandas/Shapely no disponibles → omito Shapefile. Instala con:")

        print("        pip install geopandas shapely pyogrio")

    # GeoJSON mínimo sin dependencias (fallback)

    try:

        geojson\_path = os.path.join(out\_dir, f"{name\_prefix}.geojson")

        if not os.path.exists(geojson\_path):

            coords\_ll = [[float(lon), float(lat)] for lat, lon in zip(dfp["lat"], dfp["lon"])]

            gj = {

                "type": "FeatureCollection",

                "features": [{

                    "type": "Feature",

                    "properties": {"name": name\_prefix},

                    "geometry": {"type": "LineString", "coordinates": coords\_ll}

                }]

            }

            with open(geojson\_path, "w", encoding="utf-8") as f:

                json.dump(gj, f, ensure\_ascii=False, indent=2)

            print(f"🌐  GeoJSON guardado (modo simple): {geojson\_path}")

    except Exception as e:

        print(f"[Aviso] No se pudo crear GeoJSON simple ({e}).")

    # JSON con puntos

    try:

        json\_path = os.path.join(out\_dir, f"{name\_prefix}.json")

        cols = [c for c in ["lat", "lon", "date", "time", "dist\_m"] if c in dfp.columns]

        payload = {"name": name\_prefix, "crs": "EPSG:4326", "points": dfp[cols].to\_dict(orient="records")}

        with open(json\_path, "w", encoding="utf-8") as f:

            json.dump(payload, f, ensure\_ascii=False, indent=2)

        print(f"📄  JSON guardado: {json\_path}")

    except Exception as e:

        print(f"[Aviso] No se pudo crear JSON ({e}).")

# ---------- Exportar navegación en formato ODC ----------

def export\_odc\_from\_positions(df\_pos: pd.DataFrame, out\_dir: str, arch\_dir: str, base\_name: str = "20190817112135\_v7"):

    dfp = df\_pos.dropna(subset=["lat", "lon"]).copy()

    if dfp.empty:

        print("[Aviso] No hay puntos para crear ODC.")

        return

    if "date" not in dfp.columns or "time" not in dfp.columns:

        t0 = datetime(2019, 8, 17, 11, 21, 35)

        ts = [t0 + timedelta(seconds=i) for i in range(len(dfp))]

        dfp["date"] = [dt.strftime("%Y/%m/%d") for dt in ts]

        dfp["time"] = [dt.strftime("%H:%M:%S") for dt in ts]

    lines = [f"0,151,{d},{t},{la:.6f},{lo:.6f}" for d,t,la,lo in zip(dfp["date"], dfp["time"], dfp["lat"], dfp["lon"])]

    out\_path  = os.path.join(out\_dir,  f"{base\_name}.odc")

    arch\_path = os.path.join(arch\_dir, f"{base\_name}.odc")

    with open(out\_path, "w", encoding="utf-8", newline="\n") as f:

        f.write("\n".join(lines) + "\n")

    print(f"🧭 ODC generado: {out\_path}")

    try:

        with open(arch\_path, "w", encoding="utf-8", newline="\n") as f:

            f.write("\n".join(lines) + "\n")

        print(f"🧭 ODC copiado a archivos: {arch\_path}")

    except Exception as e:

        print(f"[Aviso] No se pudo escribir ODC en ARCH\_DIR ({e}).")

# ---------- Checksums y HTML índice ----------

def write\_checksums(filepaths, out\_dir: str):

    chk\_path = os.path.join(out\_dir, "checksums.sha256")

    with open(chk\_path, "w", encoding="utf-8") as fout:

        for p in filepaths:

            try:

                h = hashlib.sha256()

                with open(p, "rb") as f:

                    for chunk in iter(lambda: f.read(1<<20), b""):

                        h.update(chunk)

                fout.write(f"{h.hexdigest()}  {os.path.basename(p)}\n")

            except Exception:

                pass

    print(f"🔏 Checksums: {chk\_path}")

def write\_index\_html(out\_dir: str, entries: dict):

    idx = os.path.join(out\_dir, "index.html")

    def li(path):

        fn = os.path.basename(path)

        return f'<li><a href="{fn}" target="\_blank">{fn}</a></li>'

    html = ["<html><head><meta charset='utf-8'><title>L35 Outputs</title></head><body>",

            "<h1>Productos L35</h1>"]

    for title, files in entries.items():

        html.append(f"<h2>{title}</h2><ul>")

        for f in files:

            if f and os.path.exists(f): html.append(li(f))

        html.append("</ul>")

    html.append("</body></html>")

    with open(idx, "w", encoding="utf-8") as f:

        f.write("\n".join(html))

    print(f"📑 Índice HTML: {idx}")

# ---------- Mapa interactivo (Folium) ----------

def make\_track\_map(df\_pos: pd.DataFrame, out\_dir: str):

    if df\_pos.empty or df\_pos["lat"].isna().all() or df\_pos["lon"].isna().all():

        print("[Aviso] Sin puntos válidos para el mapa.")

        return

    lat = df\_pos["lat"].values

    lon = df\_pos["lon"].values

    center = [float(np.nanmean(lat)), float(np.nanmean(lon))]

    m = folium.Map(location=center, zoom\_start=13, tiles="OpenStreetMap")

    coords = list(zip(lat, lon))  # (lat, lon)

    folium.PolyLine(coords, color="blue", weight=3, opacity=0.8,

                    tooltip="Línea L35 (limpia)").add\_to(m)

    folium.Marker(coords[0], popup="Inicio", icon=folium.Icon(color="green")).add\_to(m)

    folium.Marker(coords[-1], popup="Fin", icon=folium.Icon(color="red")).add\_to(m)

    out\_html = os.path.join(out\_dir, "L35\_track\_master.html")

    m.save(out\_html)

    print(f"🌍 Mapa interactivo:\n  {out\_html}")

    \_open\_file(out\_html)

# ---------------- Orquestador ----------------

def main():

    print("→ Preparando entorno…")

    os.makedirs(OUT\_DIR, exist\_ok=True)

    # 1) Leer SEG (necesario para n\_traces)

    if not os.path.exists(SEG\_PATH):

        print(f"[ERROR] No se encuentra SEG: {SEG\_PATH}")

        sys.exit(1)

    st\_tmp = read(SEG\_PATH, format="SEGY", unpack\_trace\_headers=True)

    n\_traces = len(st\_tmp)

    print(f"   SEG ok | Trazas: {n\_traces}")

    # 2) Navegación: CSV > ODC > sintetizar

    print("→ Buscando posiciones…")

    df\_csv = parse\_csv\_positions(CSV\_PATH)

    if not df\_csv.empty:

        df\_pos = df\_csv

        print(f"   Usando CSV ({len(df\_pos)} filas válidas): {CSV\_PATH}")

    else:

        df\_odc = parse\_odc\_positions(ODC\_PATH)

        if not df\_odc.empty:

            df\_pos = df\_odc

            print(f"   Usando ODC ({len(df\_pos)} filas válidas): {ODC\_PATH}")

        else:

            print("   No hay CSV/ODC válidos → sintetizando navegación…")

            df\_pos = synthesize\_navigation(n\_traces)

    # 3) Distancias y limpieza

    dist\_m, df\_pos\_clean = latlon\_to\_distance(df\_pos, n\_traces)

    if dist\_m is None:

        print("[ERROR] No se pudo calcular distancia.")

        sys.exit(1)

    total = dist\_m[-1] - dist\_m[0]

    print(f"   Distancia total (filtrada): {total:.1f} m | Trazas: {n\_traces}")

    # 4) Guardar navegación limpia

    out\_csv = os.path.join(OUT\_DIR, "L35\_rangos\_odc.csv")

    df\_pos\_clean.to\_csv(out\_csv, index=False)

    print(f"📝 L35\_rangos\_odc.csv guardado: {out\_csv}")

    # 5) Calcular UTM de inicio/fin para anotar en el perfil

    tr\_utm = Transformer.from\_crs("EPSG:4326", "EPSG:32718", always\_xy=True)

    e1, n1 = tr\_utm.transform(float(df\_pos\_clean["lon"].iloc[0]), float(df\_pos\_clean["lat"].iloc[0]))

    e2, n2 = tr\_utm.transform(float(df\_pos\_clean["lon"].iloc[-1]), float(df\_pos\_clean["lat"].iloc[-1]))

    # 6) Perfil + Mapa + Vectoriales

    plot\_profile\_with\_distance(SEG\_PATH, dist\_m, OUT\_DIR, utm\_start\_end=(e1, n1, e2, n2))

    try:

        make\_track\_map(df\_pos\_clean, OUT\_DIR)

    except Exception as e:

        print(f"[Aviso] No se pudo generar el mapa (continuo con exportaciones): {e}")

    export\_track\_products(df\_pos\_clean, OUT\_DIR, name\_prefix="L35\_track")

    # 7) ODC derivado (y copia a ARCH\_DIR)

    export\_odc\_from\_positions(df\_pos\_clean, OUT\_DIR, ARCH\_DIR, base\_name="20190817112135\_v7")

    # 8) Índice HTML y checksums

    files = [

        os.path.join(OUT\_DIR, "L35\_profile\_master.jpg"),

        os.path.join(OUT\_DIR, "L35\_profile\_master.svg"),

        os.path.join(OUT\_DIR, "L35\_trace\_distance\_map.csv"),

        os.path.join(OUT\_DIR, "L35\_rangos\_odc.csv"),

        os.path.join(OUT\_DIR, "L35\_track.kmz"),

        os.path.join(OUT\_DIR, "L35\_track.kml"),

        os.path.join(OUT\_DIR, "L35\_track.geojson"),

        os.path.join(OUT\_DIR, "L35\_track.json"),

        os.path.join(OUT\_DIR, "20190817112135\_v7.odc"),

        os.path.join(OUT\_DIR, "L35\_track.shp"),

    ]

    write\_checksums([p for p in files if os.path.exists(p)], OUT\_DIR)

    write\_index\_html(OUT\_DIR, {

        "Perfil": [os.path.join(OUT\_DIR, "L35\_profile\_master.jpg"),

                   os.path.join(OUT\_DIR, "L35\_profile\_master.svg")],

        "Navegación / QA": [os.path.join(OUT\_DIR, "L35\_rangos\_odc.csv"),

                            os.path.join(OUT\_DIR, "L35\_trace\_distance\_map.csv"),

                            os.path.join(OUT\_DIR, "20190817112135\_v7.odc")],

        "Vectoriales": [os.path.join(OUT\_DIR, "L35\_track.kmz"),

                        os.path.join(OUT\_DIR, "L35\_track.kml"),

                        os.path.join(OUT\_DIR, "L35\_track.geojson"),

                        os.path.join(OUT\_DIR, "L35\_track.shp")],

        "Mapa interactivo": [os.path.join(OUT\_DIR, "L35\_track\_master.html")],

        "Otros": [os.path.join(OUT\_DIR, "checksums.sha256")]

    })

if \_\_name\_\_ == "\_\_main\_\_":

    main()

### README — Ejecución rápida L35 (V7)

==================================

Proyecto: CHIRPVIEW (V7) / Línea L35

Script base: C:\Python2025\_SAC\GITHUBL58\src\L35masterV7.py

Lanzador: C:\Python2025\_SAC\GITHUBL58\run\_L35masterV7.bat

1) Requisitos mínimos

---------------------

- Windows 10/11

- Python 3.10+ (recomendado)

- Entorno virtual del proyecto en: C:\Python2025\_SAC\GITHUBL58\.venv

(opcional pero recomendado)

2) Estructura de carpetas (esperada)

------------------------------------

C:\Python2025\_SAC\GITHUBL58\

├─ .venv\

├─ Handbook\

├─ LinesDat.L35\

│ ├─ 20190817112135.seg (obligatorio)

│ ├─ 20190817112135.odc (opcional)

│ └─ 20190817112135.csv (opcional)

├─ out\L35\ (se crea automáticamente)

├─ src\

│ └─ L35masterV7.py

└─ run\_L35masterV7.bat

3) Cómo ejecutar

----------------

Opción A — Doble clic:

- Hacer doble clic en: run\_L35masterV7.bat

Opción B — Desde PowerShell:

- Abrir PowerShell

- Ejecutar:

cd "C:\Python2025\_SAC\GITHUBL58"

.\run\_L35masterV7.bat

(El BAT intenta usar el Python del venv: .venv\Scripts\python.exe.

Si no lo encuentra, usa el Python del sistema.)

4) Qué genera (salida)

----------------------

Se guardan en: C:\Python2025\_SAC\GITHUBL58\out\L35\

- L35\_profile\_master.jpg / .svg (perfil sísmico con distancia y profundidad)

- L35\_track\_master.html (mapa interactivo Folium)

- L35\_track.kml / .kmz (vectorial para Google Earth)

- L35\_track.shp (Shapefile; requiere GeoPandas/Shapely)

- L35\_track.geojson (GeoJSON; se genera aun sin GeoPandas)

- L35\_track.json (coordenadas y QA en JSON)

- L35\_rangos\_odc.csv (navegación limpia + distancia)

- L35\_trace\_distance\_map.csv (traza vs. distancia)

- 20190817112135\_v7.odc (ODC generado)

- checksums.sha256 (verificación)

- index.html (índice de productos)

5) Log y errores

----------------

- Log de ejecución: C:\Python2025\_SAC\GITHUBL58\out\L35\run\_L35masterV7.log

- Si el BAT muestra “[ERROR]”, abrir el log y revisar:

\* “No se encuentra SEG” → verifique que 20190817112135.seg exista en LinesDat.L35

\* “No se pudo escribir Shapefile/GeoJSON” → instale dependencias Geo (ver punto 7)

\* Sin CSV/ODC → el script sintetiza navegación automáticamente (modo demo)

- Atajo (opcional) para abrir el log:

notepad "C:\Python2025\_SAC\GITHUBL58\out\L35\run\_L35masterV7.log"

6) Parámetros clave (solo si necesita ajustar)

----------------------------------------------

- Proyección UTM: 18S (EPSG:32718)

- Velocidad del sonido VEL\_MS = 1500.0 m/s

- Escala vertical Z\_MULT = 10.0 (1.0 = sin estirar)

- Paleta del perfil: CMAP = "turbo"

- Etiquetas UTM (inicio/fin) debajo del eje X, en negro, tamaño grande.

7) Dependencias opcionales para Shapefile (SHP)

-----------------------------------------------

Para exportar .shp con GeoPandas/Shapely (GeoJSON no es obligatorio):

- Activar venv:

cd C:\Python2025\_SAC\GITHUBL58

.\.venv\Scripts\activate

- Instalar:

pip install geopandas shapely pyogrio

8) Preguntas rápidas

--------------------

- ¿Puedo ejecutar sin CSV/ODC? Sí. El script genera navegación sintética.

- ¿Dónde veo todo de una vez? Abra out\L35\index.html

- ¿Cómo cambio colores o escala? Editar CMAP y Z\_MULT en src\L35masterV7.py

Contacto / Notas

----------------

Cualquier cambio futuro debe partir desde la versión base: L35masterV7.py.

Este README es un instructivo mínimo para usuarios finales (doble clic en el BAT).