Manual Técnico

Toolbox Científica en Python
para Análisis Experimental

Documentación Técnica

31 de enero de 2026

Índice general

Introducción General

Qué es esta toolbox

Esta toolbox es una colección modular de herramientas Python diseñadas específicamente para el análisis de datos experimentales en física. No pretende ser una reimplementación de NumPy, SciPy o pandas, sino una capa semántica orientada al flujo de trabajo experimental: desde la adquisición de datos, pasando por el análisis estadístico, el ajuste de modelos, hasta la presentación metrológica de resultados con incertidumbres.

El diseño responde a una necesidad concreta: reducir la fricción entre el *razonamiento físico-matemático* y su implementación computacional. Cada módulo está concebido para reflejar una distinción conceptual clara, evitando la mezcla indiscriminada de responsabilidades que a menudo lleva a código confuso o resultados mal interpretados.

Filosofía general: separación de capas

La arquitectura de la toolbox se basa en una separación estricta de capas funcionales:

1. **Cálculo numérico puro (NumPy/SciPy)**: Operaciones algebraicas, vectoriales, matriciales. No añade semántica física.

2. **Estadística inferencial (`estadistica.py`)**: Extracción de información a partir de datos observados. Se enfoca exclusivamente en incertidumbre Tipo A según ISO GUM (análisis estadístico de observaciones repetidas). Devuelve valores escalares (`float`), intervalos, o diccionarios con resultados de tests. **No maneja representaciones tipo "valor ± incertidumbre"**.

3. **Ajuste de modelos (`ajustes.py`)**: Estimación de parámetros mediante regresión (mínimos cuadrados, máxima verosimilitud, etc.). Devuelve parámetros óptimos y matrices de covarianza. No calcula directamente incertidumbres combinadas ni propagaciones simbólicas.

4. **Metrología e incertidumbres (`incertidumbres.py`)**: Capa de frontera. Construye magnitudes con incertidumbre (usando la librería `uncertainties`), combina incertidumbres Tipo A y Tipo B, propaga incertidumbres a través de cálculos, y formatea resultados para LaTeX. Este módulo *no calcula estadística ni ajustes*, solo los representa y combina.

5. **Visualización y presentación (`graficos.py`, `latex_tools.py`)**: Generación de gráficos y tablas con formato consistente.

Principio rector: Cada módulo debe ser conceptualmente coherente. Si una función mezcla estadística con propagación de incertidumbres instrumentales, probablemente está en el módulo equivocado.

Mapa de módulos

A continuación se presenta un resumen de los módulos principales (algunos aún por documentar en este manual):

- `estadistica.py` (Capítulo **??**): Estadística descriptiva, intervalos de confianza, tests de hipótesis. Incertidumbre Tipo A.
- `ajustes.py` (Capítulo futuro): Regresión lineal y no lineal, mínimos cuadrados, estimación robusta.
- `incertidumbres.py` (Capítulo futuro): Construcción de magnitudes con incertidumbre, propagación, combinación Tipo A + Tipo B.
- `montecarlo.py` (Capítulo futuro): Simulación Monte Carlo para propagación de incertidumbres complejas.
- `graficos.py` (Capítulo futuro): Visualización de datos experimentales con estilo consistente.
- `latex_tools.py` (Capítulo futuro): Generación automatizada de tablas y expresiones LaTeX.

En este documento, nos centraremos en el módulo `estadistica.py`, por ser la base conceptual sobre la que se construyen los demás.

Capítulo 1

Estadística (`estadistica.py`)

1.1. Rol del módulo de estadística

1.1.1. Qué problemas resuelve

El módulo `estadistica.py` está diseñado para resolver problemas de **inferencia estadística a partir de datos experimentales**. En el contexto de la física experimental, esto incluye:

- Estimar el valor "verdadero" de una magnitud a partir de mediciones repetidas (media muestral).
- Cuantificar la dispersión de las observaciones (varianza, desviación típica).
- Estimar la incertidumbre del valor central debido a la variabilidad estadística (error estándar).
- Construir intervalos de confianza para parámetros poblacionales.
- Contrastar hipótesis sobre los datos (tests de significancia).

Todas estas operaciones corresponden a lo que la *Guía para la Expresión de la Incertidumbre de Medida* (ISO GUM) clasifica como **incertidumbre Tipo A**: evaluada mediante análisis estadístico de series de observaciones.

1.1.2. Qué problemas NO pretende resolver

Este módulo **no** aborda:

- **Incertidumbre Tipo B**: Evaluación de incertidumbres basada en información externa (resolución del instrumento, certificados de calibración, juicio experto). Esto se maneja en `incertidumbres.py`.
- **Propagación de incertidumbres**: Cálculo de cómo las incertidumbres de las variables de entrada afectan a una función. Esto también corresponde a `incertidumbres.py`.
- **Ajuste de modelos**: Estimación de parámetros de funciones teóricas mediante regresión. Esto se trata en `ajustes.py`.
- **Representación metrológica**: El módulo devuelve valores escalares (`float`), no magnitudes tipo "valor ± incertidumbre". La construcción de objetos `ufloat` es responsabilidad de `incertidumbres.py`.

1.1.3. Relación con otros módulos

El flujo típico de trabajo es:

1. **Recolección de datos**: Arrays NumPy con mediciones repetidas.
2. **Análisis estadístico (`estadistica.py`)**: Cálculo de media, error estándar, intervalos de confianza. Salida: valores `float`.
3. **Ajuste de modelos (`ajustes.py`)**: Si los datos siguen un modelo teórico, se estiman parámetros y su matriz de covarianza. Salida: parámetros `float` + matriz de covarianza.
4. **Metrología (`incertidumbres.py`)**: Se combinan incertidumbres Tipo A (de `estadistica.py` o `ajustes.py`) con Tipo B (instrumentales). Se construyen objetos `ufloat` para propagación automática.
5. **Presentación**: Generación de tablas, gráficos y expresiones LaTeX.

Importante: `estadistica.py` es una capa de cálculo puro. No debe importar `uncertainties` ni devolver `ufloat`. La semántica metrológica se añade *después*, en la capa correspondiente.

1.2. Estadística descriptiva

1.2.1. Media muestral

Definición 1.2.1 (Media muestral)**.** Dada una muestra $\{x_1, x_2, \ldots, x_n\}$ de n observaciones independientes de una variable aleatoria X, la media muestral se define como:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1.1}$$

Interpretación física: La media muestral es el mejor estimador (no sesgado, varianza mínima) del valor esperado $\mu = \mathbb{E}[X]$ de la magnitud subyacente, bajo el supuesto de que las observaciones son independientes y provienen de la misma distribución.

Implementación:

```
from estadistica import estadistica
import numpy as np

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
media = estadistica.media(x)  # Devuelve: float
```

Consideraciones:

- La media es sensible a valores atípicos (outliers). Si la distribución es fuertemente asimétrica o presenta colas pesadas, considerar usar la mediana.
- Para distribuciones gaussianas, la media muestral es el estimador de máxima verosimilitud de μ.

1.2.2. Varianza y desviación típica

Definición 1.2.2 (Varianza muestral)**.** La varianza muestral (corregida por sesgo) se define como:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{1.2}$$

donde el factor $n-1$ (en lugar de n) corrige el sesgo del estimador, aplicando la corrección de Bessel.

Definición 1.2.3 (Desviación típica muestral)**.** La desviación típica (o estándar) muestral es la raíz cuadrada de la varianza:

$$s = \sqrt{s^2} \tag{1.3}$$

Interpretación física: s cuantifica la dispersión de las observaciones respecto a su media. Es un estimador no sesgado de la desviación típica poblacional σ si las observaciones son independientes y provienen de una distribución normal.

Implementación:

```
varianza = estadistica.varianza(x, ddof=1)  # ddof=1: muestra
sigma = estadistica.desviacion_tipica(x, ddof=1)
```

Parámetro `ddof`:

- `ddof=1`: Varianza muestral (divide por $n-1$). Usado cuando x es una muestra.
- `ddof=0`: Varianza poblacional (divide por n). Usado cuando x es la población completa.

En física experimental, casi siempre se usa `ddof=1`, ya que trabajamos con muestras.

1.2.3. Error estándar de la media

Definición 1.2.4 (Error estándar de la media)**.** El error estándar de la media (standard error of the mean, SEM) cuantifica la incertidumbre del estimador $\bar{x}$ debido a la variabilidad estadística:

$$\sigma_{\bar{x}} = \frac{s}{\sqrt{n}} \tag{1.4}$$

Interpretación física: Mientras que s describe la dispersión de las observaciones individuales, $\sigma_{\bar{x}}$ describe cuánto esperamos que $\bar{x}$ se desvíe del valor verdadero μ debido al tamaño finito de la muestra. Es directamente la **incertidumbre Tipo A** de la media.

Relación con intervalos de confianza: El error estándar es el ingrediente fundamental para construir intervalos de confianza para μ (ver sección **??**).

Implementación:

```
error_std = estadistica.error_estandar(x)
# Devuelve: sigma_x_barra = s / sqrt(n)
```

Observación 1.2.1. El error estándar disminuye como $1/\sqrt{n}$. Duplicar la precisión requiere cuadruplicar el número de mediciones. Esto es fundamental para planificar experimentos.

1.2.4. Covarianza y correlación

Para dos variables X e Y:

Definición 1.2.5 (Covarianza muestral)**.**

$$\mathrm{Cov}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \tag{1.5}$$

Definición 1.2.6 (Coeficiente de correlación de Pearson)**.**

$$r = \frac{\mathrm{Cov}(X,Y)}{s_X\, s_Y} \tag{1.6}$$

donde $r \in [-1, 1]$.

Interpretación: $r = 1$ indica correlación lineal perfecta positiva, $r = -1$ correlación perfecta negativa, $r = 0$ ausencia de correlación lineal (pero no necesariamente independencia).

Implementación:

```
cov_xy = estadistica.covarianza(x, y, ddof=1)
r = estadistica.correlacion(x, y)
```

Uso en propagación de incertidumbres: La covarianza entre variables es esencial para propagar correctamente incertidumbres cuando las variables están correlacionadas. Esto se usará en `ajustes.py` (matriz de covarianza de parámetros) y en `incertidumbres.py` (propagación con correlaciones).

1.3. Intervalos de confianza

1.3.1. Diferencia entre incertidumbre e intervalo de confianza

Es crucial distinguir:

- **Incertidumbre (error estándar)**: Cuantifica la dispersión esperada del estimador. Es un número: $\sigma_{\bar{x}} = s/\sqrt{n}$.
- **Intervalo de confianza**: Es un intervalo $[L, U]$ construido de tal forma que, si repitiéramos el experimento muchas veces, el parámetro verdadero μ estaría dentro del intervalo en un porcentaje de las veces igual al *nivel de confianza* (típicamente 95 %).

Interpretación correcta: Un intervalo de confianza al 95 % significa que si repitiéramos el procedimiento experimental (incluyendo muestreo y cálculo del intervalo) infinitas veces, en el 95 % de los casos el intervalo contendría el valor verdadero μ. **No significa** que hay 95 % de probabilidad de que μ esté en ese intervalo particular (eso sería interpretación bayesiana, no frecuentista).

1.3.2. Intervalo para la media con σ conocida

Si conocemos la desviación típica poblacional σ (situación poco común en la práctica), y asumimos normalidad de los datos, el intervalo de confianza al nivel $(1 - \alpha)$ para μ es:

$$\bar{x} \pm z_{\alpha/2} \frac{\sigma}{\sqrt{n}} \tag{1.7}$$

donde $z_{\alpha/2}$ es el cuantil de la distribución normal estándar que deja probabilidad $\alpha/2$ en cada cola.

Implementación:

```
li, ls = estadistica.intervalo_media_sigma_conocida(
    x, sigma=0.5, nivel=0.95
)
# Devuelve: (limite_inferior, limite_superior)
```

1.3.3. Intervalo para la media con σ desconocida (t-Student)

En la práctica, no conocemos σ y lo estimamos con s. En este caso, la distribución del estadístico:

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}} \tag{1.8}$$

sigue una distribución t de Student con $n - 1$ grados de libertad.

El intervalo de confianza al nivel $(1-\alpha)$ es:

$$\bar{x} \pm t_{\alpha/2,n-1} \frac{s}{\sqrt{n}} \tag{1.9}$$

donde $t_{\alpha/2,n-1}$ es el cuantil de la distribución t con $n-1$ grados de libertad.
Implementación:

```
li, ls = estadistica.intervalo_media_sigma_desconocida(
    x, nivel=0.95
)
```

Observaciones:

- Para n grande ($n > 30$), la distribución t converge a la normal estándar.
- Para muestras pequeñas, t tiene colas más pesadas que la normal, reflejando la incertidumbre adicional por estimar σ.
- **Supuesto crítico**: Los datos deben provenir (aproximadamente) de una distribución normal. Para datos muy asimétricos o con outliers, el intervalo puede no ser válido.

1.3.4. Intervalo para la varianza

El intervalo de confianza para σ^2 (asumiendo normalidad) se basa en la distribución χ^2:

$$\left[\frac{(n-1)s^2}{\chi^2_{\alpha/2,n-1}}, \frac{(n-1)s^2}{\chi^2_{1-\alpha/2,n-1}} \right] \tag{1.10}$$

Implementación:

```
li_var, ls_var = estadistica.intervalo_varianza(x, nivel=0.95)
```

Advertencia: Este intervalo es mucho más sensible a desviaciones de la normalidad que el intervalo para la media. Usar con precaución.

1.4. Tests estadísticos

1.4.1. Hipótesis nula y alternativa

Un test de hipótesis contrasta dos afirmaciones mutuamente excluyentes:

- **Hipótesis nula (H_0)**: Afirmación que se asume cierta por defecto (p.ej., "no hay efecto", "la media es μ_0").
- **Hipótesis alternativa (H_1 o H_a)**: Afirmación que se acepta si hay evidencia suficiente contra H_0.

El procedimiento:

1. Calcular un **estadístico de prueba** a partir de los datos.
2. Determinar la distribución del estadístico bajo H_0.
3. Calcular el **p-valor**: probabilidad de observar un estadístico tan extremo (o más) que el observado, si H_0 fuera cierta.
4. Si $p < \alpha$ (nivel de significancia, típicamente 0.05), se rechaza H_0.

1.4.2. Interpretación del p-valor

El p-valor NO es:

- La probabilidad de que H_0 sea cierta.
- La probabilidad de haber cometido un error.

El p-valor SÍ es:

- La probabilidad de obtener datos tan extremos como los observados (o más), *asumiendo que H_0 es cierta*.
- Una medida de la *compatibilidad* entre los datos y H_0.

Un p-valor pequeño indica que los datos son poco probables bajo H_0, sugiriendo evidencia en contra de H_0. Pero **no prueba** que H_1 sea cierta, ni cuantifica su probabilidad.

1.4.3. Test t de Student

Objetivo: Contrastar si la media poblacional μ es igual a un valor hipotético μ_0.
Hipótesis:

$$H_0 : \mu = \mu_0 \tag{1.11}$$
$$H_1 : \mu \neq \mu_0 \quad \text{(dos colas)} \tag{1.12}$$

Estadístico de prueba:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \tag{1.13}$$

Bajo H_0, t sigue una distribución t de Student con $n - 1$ grados de libertad.
Implementación:

```
resultado = estadistica.test_media_t(
    x, mu0=3.0, alternativa="dos_colas"
)
# Devuelve: {"t": estadistico, "p": p_valor, "df": grados_libertad}
print(f"t = {resultado['t']:.3f}, p = {resultado['p']:.4f}")
```

Alternativas:

- `alternativa="dos_colas"`: $H_1 : \mu \neq \mu_0$ (más común).
- `alternativa="mayor"`: $H_1 : \mu > \mu_0$ (test unilateral).
- `alternativa="menor"`: $H_1 : \mu < \mu_0$ (test unilateral).

Interpretación:

- Si $p < 0{,}05$: Rechazamos H_0 al nivel de significancia del 5 %. Hay evidencia de que $\mu \neq \mu_0$.
- Si $p \geq 0{,}05$: No rechazamos H_0. No hay evidencia suficiente para descartar que $\mu = \mu_0$.

Supuestos:

- Observaciones independientes.
- Distribución aproximadamente normal (menos crítico para n grande por el Teorema Central del Límite).

1.4.4. Test de Kolmogórov–Smirnov

Objetivo: Contrastar si una muestra proviene de una distribución teórica especificada.
Hipótesis:

$$H_0 : \text{Los datos siguen la distribución teórica } F_0 \tag{1.14}$$
$$H_1 : \text{Los datos no siguen } F_0 \tag{1.15}$$

Estadístico de prueba: Máxima distancia entre la función de distribución empírica y la teórica:

$$D = \sup_x |F_n(x) - F_0(x)| \tag{1.16}$$

Implementación:

```
resultado = estadistica.test_ks(x, distribucion="normal")
# Devuelve: {"estadistico": D, "p_valor": p}
print(f"D = {resultado['estadistico']:.3f}, p = {resultado['p_valor']:.4
    f}")
```

Distribuciones soportadas:

- `"normal"`: Estima μ y σ de los datos.
- `üniforme"`: Distribución uniforme estándar.

Advertencias:

- Si los parámetros de la distribución teórica se estiman de los datos (como en `"normal"`), el test es conservador (tiende a no rechazar H_0).
- El test K-S es sensible a diferencias en toda la distribución, no solo en las colas o la media.
- Para contrastar normalidad con parámetros estimados, el test de Shapiro-Wilk es generalmente más potente.

1.5. Conexión con otros módulos

1.5.1. Salidas hacia `ajustes.py`

El módulo de ajustes utilizará frecuentemente:

- **Varianza residual**: Para estimar la bondad del ajuste.
- **Correlación**: Para detectar correlaciones entre variables explicativas.
- **Error estándar**: Como peso en regresiones ponderadas.

La filosofía es la misma: `ajustes.py` devuelve parámetros óptimos y matriz de covarianza (valores `float`), sin construir `ufloat`.

1.5.2. Salidas hacia `incertidumbres.py`

El módulo de incertidumbres combinará:

- **Incertidumbre Tipo A**: Error estándar calculado por `estadistica.py`.
- **Incertidumbre Tipo B**: Información instrumental o externa (resolución, calibración).

Ejemplo de flujo típico:

```python
from estadistica import estadistica
from incertidumbres import incertidumbres

# 1. Mediciones repetidas
x = np.array([1.0, 1.1, 0.9, 1.05, 0.95])

# 2. Estadistica Tipo A (estadistica.py)
x_media = estadistica.media(x)
u_A = estadistica.error_estandar(x)

# 3. Incertidumbre Tipo B (externa)
u_B = 0.02  # Resolucion del instrumento

# 4. Combinacion (incertidumbres.py)
x_final = incertidumbres.u(x_media, np.sqrt(u_A**2 + u_B**2))
# x_final es un ufloat, listo para propagacion
```

1.5.3. Qué NO debe hacerse en `estadistica.py`

- **No** devolver objetos `ufloat`.
- **No** combinar incertidumbres Tipo A y Tipo B.
- **No** implementar propagación simbólica de errores.
- **No** ajustar modelos (eso es responsabilidad de `ajustes.py`).

Mantener esta separación permite que cada módulo sea conceptualmente claro, testeable de forma independiente, y reutilizable en diferentes contextos.

1.6. Conclusión del capítulo

El módulo `estadistica.py` proporciona las herramientas fundamentales para la **inferencia estadística a partir de datos experimentales**, enfocándose exclusivamente en incertidumbre Tipo A según la clasificación ISO GUM.

Sus funciones devuelven valores escalares (`float`), intervalos, o diccionarios con resultados de tests, sin introducir representaciones metrológicas tipo "valor ± incertidumbre". Esta pureza conceptual facilita:

- **Claridad**: Cada módulo tiene un propósito bien delimitado.
- **Testabilidad**: Funciones puras son más fáciles de validar.
- **Composición**: Los resultados de `estadistica.py` se combinan naturalmente con otros módulos.

En los siguientes capítulos (futuros), exploraremos cómo los resultados de `estadistica.py` se integran con:

- **Ajuste de modelos (`ajustes.py`)**: Estimación de parámetros y sus incertidumbres correlacionadas.
- **Metrología (`incertidumbres.py`)**: Combinación de Tipo A + Tipo B, propagación automática, formateo para presentación.

Esta arquitectura modular refleja la estructura conceptual del análisis experimental moderno, reduciendo la fricción entre el razonamiento físico y su implementación computacional.

Capítulo 2

Generación de resultados en LaTeX: `latex_tools.py`

2.1. Rol del módulo de presentación

2.1.1. Contexto en el flujo experimental

El módulo `latex_tools.py` completa el flujo experimental:

Experimento → Adquisición (Excel/CSV) → Python: `estadistica.py`;

→ `ajustes.py` → `incertidumbres.py` → `latex_tools.py` → LaTeX → PDF/Artículo

Propósito específico: Transformar resultados numéricos (valores con incertidumbre) en código LaTeX de presentación profesional, listo para compilar en documentos científicos.

Qué NO hace este módulo:

- No calcula estadística (eso es `estadistica.py`).
- No combina incertidumbres Tipo A + Tipo B (eso es `incertidumbres.py`).
- No propaga incertidumbres automáticamente.
- No ajusta modelos (eso es `ajustes.py`).

Qué hace este módulo:

- Redondea valores e incertidumbres siguiendo normas metrológicas.
- Formatea valores escalares con su incertidumbre como expresiones LaTeX.
- Transforma arrays/matrices en tablas LaTeX estilizadas.
- Maneja unidades físicas y soporte `siunitx`.
- Exporta contenido LaTeX a archivos `.tex`.

2.1.2. Separación de responsabilidades

Es crucial entender que **presentación ≠ cálculo**:

- **Cálculo**: NumPy → `estadistica.py` (float/ndarray) → `incertidumbres.py` (ufloat).

- **Presentación**: `latex_tools.py` toma valores ya calculados (float o ufloat) y los transforma en código LaTeX.

Esto permite que `latex_tools.py` sea **independiente de la fuente de los datos**. Puede recibir valores de `estadistica.py`, `ajustes.py`, simulaciones Monte Carlo, o cálculos simbólicos. El módulo no pregunta "de dónde vienes"; solo "¿cuál es tu valor y tu incertidumbre?".

2.2. Redondeo metrológico

2.2.1. Principios de redondeo

El redondeo de magnitudes con incertidumbre sigue la norma ISO GUM y la práctica metrológica estándar:

Definición 2.2.1 (Regla de redondeo metrológico)**.** Dado un valor x con incertidumbre u (o σ):

1. La incertidumbre se expresa con 1 o 2 cifras significativas (típicamente 2 en equipos de precisión).
2. El valor se redondea al mismo orden decimal que la incertidumbre.
3. El orden decimal se determina por el orden de magnitud decimal de la incertidumbre.

Ejemplo 2.2.1 (Ejemplo numérico)**.** Supongamos $x = 9{,}8127$ m/s y $\sigma = 0{,}0347$ m/s:
Paso 1: Calcular el orden de magnitud decimal de σ:

$$\log_{10}(0{,}0347) \approx -1{,}46 \implies \text{orden} = -2$$

Paso 2: Con 2 cifras significativas en la incertidumbre:

$$\text{decimales} = -((-2) - (2-1)) = -(-1) = 1$$

Redondeamos σ a 1 decimal:

$$\sigma_r = \text{round}(0{,}0347, 1) = 0{,}0$$

Problema: 0.0 es demasiado pequeño. Generalmente se usan 2 cifras significativas:

$$\sigma_r = \text{round}(0{,}0347, 2) = 0{,}03$$

Paso 3: Redondeamos x al mismo decimal:

$$x_r = \text{round}(9{,}8127, 2) = 9{,}81$$

Resultado final: $(9{,}81 \pm 0{,}03)$ m/s

2.2.2. La función `redondeo_incertidumbre()`

```
redondeo_incertidumbre(valor, sigma, cifras=2)
    -> (valor_redondeado, sigma_redondeada, decimales)
```

Parámetros:

- `valor` (float): Valor central x.
- `sigma` (float): Incertidumbre $\sigma > 0$.

- `cifras` (int, default=2): Cifras significativas para σ (1 o 2).

Devuelve:

- `valor_redondeado` (float): x redondeado.
- `sigma_redondeada` (float): σ redondeada.
- `decimales` (int): Número de decimales aplicado.

Uso típico:

```
from latex_tools import latex_tools

v_r, s_r, dec = latex_tools.redondeo_incertidumbre(
    valor=9.8127,
    sigma=0.0347,
    cifras=2
)
# Devuelve: (9.81, 0.03, 2)
```

Nota: Esta función es **determinista**, no tiene efectos secundarios y puede usarse sin dependencias de estado.

2.3. Formato de valores con incertidumbre

2.3.1. La función `valor_pm()`

`valor_pm()` es la función **polivalente** del módulo. Dependiendo del tipo de entrada, cambia su comportamiento:

Definición 2.3.1. La función `valor_pm()` acepta:

- **Escalar**: Devuelve una expresión LATEX de display math.
- **Vector 1-D**: Devuelve tabla LATEX (vertical u horizontal).
- **Matriz 2-D**: Devuelve tabla LATEX rectangular.

Listing 2.1: Firma de valor_pm()

```
valor_pm(
    valor,
    sigma=None,
    *,
    unidad=None,
    cifras=2,
    siunitx=False,
    caption=None,
    label=None,
    centrar=None,
    tamano=None,
    lineas=None,
    envolver=None,
    posicion=None,
    orientacion="vertical",
    headers=None,
    row_headers=None
)
```

Parámetros clave:

- `valor`: float, array, o `uncertainties.uarray`.
- `sigma`: incertidumbre(s). Si es `None`, `valor` se interpreta como `ufloat`.
- `cifras`: 1 o 2 cifras significativas en σ.
- `unidad`: string opcional para la unidad física (ej: `"m/s^2"`).
- `siunitx`: si True, activa formato `\SI{...}{...}` (requiere `unidad`).

2.3.2. Caso escalar

Entrada:

```
latex_tools.valor_pm(9.81, 0.05, unidad="m/s^2", cifras=2)
```

Salida:

```
\[(9.81 \pm 0.05)\,\mathrm{m/s^2}\]
```

Cuando se compila en LaTeX, produce una expresión centrada en display math:

$$(9{,}81 \pm 0{,}05)\,\mathrm{m/s^2}$$

Sin unidad:

```
latex_tools.valor_pm(3.14159, 0.0123, cifras=2)
# Devuelve: \[(3.14 \pm 0.01)\]
```

Con siunitx:

```
latex_tools.valor_pm(
    9.81, 0.05,
    unidad="m/s^2",
    cifras=2,
    siunitx=True
)
# Devuelve: \SI{9.81 \pm 0.05}{m/s^2}
```

(Requiere `\usepackagesiunitx` en el preámbulo LaTeX).

2.3.3. Notación científica automática

Si el valor o la incertidumbre son muy grandes ($\geq 10^5$) o muy pequeños ($< 10^{-4}$), la función activa automáticamente notación científica:

Ejemplo 2.3.1 (Notación científica).

```
latex_tools.valor_pm(0.000123, 0.000005, cifras=2)
# Devuelve: \[(1.23 \times 10^{-4} \pm 5.0 \times 10^{-6})\]
```

Que se renderiza como:

$$(1{,}23 \times 10^{-4} \pm 5{,}0 \times 10^{-6})$$

2.3.4. Caso vectorial: tablas

Vector vertical (por defecto)

```
x = np.array([1.0, 2.0, 3.0])
sx = 0.1
tex = latex_tools.valor_pm(
    x, sx,
    cifras=1,
    caption="Medidas de posicion",
    label="tab:posicion"
)
print(tex)
```

Salida: Tabla LATEX con estructura:

```
\begin{table}[htbp]
\centering
\begin{tabular}{c}
\hline
(1.0 \pm 0.1) \\
(2.0 \pm 0.1) \\
(3.0 \pm 0.1) \\
\hline
\end{tabular}
\caption{Medidas de posición}
\label{tab:posicion}
\end{table}
```

Vector horizontal

```
tex = latex_tools.valor_pm(
    x, sx,
    cifras=1,
    orientacion="horizontal"
)
```

Genera tabla de 1 fila $\times$ 3 columnas.

Con encabezados

```
x = np.array([1.0, 2.0, 3.0])
y = np.array([10.5, 20.3, 30.1])
sx, sy = 0.1, 0.2

# Combinar en matriz 3x2
V = np.array([x, y]).T
sV = np.array([sx, sy])

tex = latex_tools.valor_pm(
    V, sV,
    cifras=1,
    headers=["$x$ (m)", "$y$ (V)"],
    caption="Datos experimentales"
)
```

Genera tabla con encabezados de columna.

2.4. Configuración global de tablas

2.4.1. El diccionario TABLA_CONFIG

La apariencia de todas las tablas puede configurarse globalmente mediante el diccionario TABLA_CONFIG:

```
TABLA_CONFIG = {
    "centrar": True,            # Centra la tabla
    "tamano": None,             # None, "small", "footnotesize", etc.
    "lineas": "hline",          # "booktabs", "hline", o None
    "envolver": True,           # Envuelve en \begin{table}...\end{table}
    "posicion": "htbp",         # Posicion del flotante
}
```

Parámetros:

- centrar: Si True, añade \centering.
- tamano: Tamaño de fuente (\small, \footnotesize, etc.). None = sin cambio.
- lineas: Estilo de líneas:
  - "hline": Líneas horizontales (clásico).
  - "booktabs": Líneas elegantes (requiere \usepackagebooktabs).
  - None: Sin líneas.
- envolver: Si True, genera \begin{table}.
- posicion: Parámetro de colocación ([H], [htbp], etc.).

Cambiar configuración global:

```
from latex_tools import latex_tools, TABLA_CONFIG

# Cambiar estilo global
TABLA_CONFIG["lineas"] = "booktabs"
TABLA_CONFIG["tamano"] = "small"

# Ahora todas las nuevas tablas usar n estos estilos
```

Sobrescribir localmente:

```
# Esta tabla especifica ignora TABLA_CONFIG
latex_tools.valor_pm(
    x, sx,
    lineas="hline",     # Sobrescribe TABLA_CONFIG["lineas"]
    tamano="large"      # Sobrescribe TABLA_CONFIG["tamano"]
)
```

2.5. Exportación a archivo

2.5.1. La función exportar()

```
exportar(filename, contenido, modo="w")
```

Propósito: Escribir contenido LATEX generado en Python a un archivo .tex.
Parámetros:

- `filename` (str): Ruta del archivo (ej: `"tablas/resultados.tex"`).
- `contenido` (str): Código LaTeX a escribir.
- `modo` (str, default=`"w"`): `"w"` (sobrescribir) o `."a"` (añadir).

Ejemplo de flujo reproducible:

```python
import numpy as np
from estadistica import estadistica
from latex_tools import latex_tools

# 1. Adquirir datos
datos_x = np.array([1.0, 1.1, 0.9, 1.05, 0.95])

# 2. Analisis estadistico
media_x = estadistica.media(datos_x)
error_x = estadistica.error_estandar(datos_x)

# 3. Formatear para LaTeX
tex_resultado = latex_tools.valor_pm(
    media_x, error_x,
    unidad="m",
    cifras=2
)

# 4. Guardar en archivo
latex_tools.exportar("resultados.tex", tex_resultado)
```

Este flujo es completamente reproducible: dado el mismo código Python, siempre produce el mismo `.tex`.

2.5.2. Nota sobre reproducibilidad

Observación 2.5.1 (Reproducibilidad científica)**.** Exportar resultados a archivos `.tex` generados automáticamente garantiza:

- **Trazabilidad**: El `.tex` es siempre derivado del código Python, nunca editado manualmente.
- **Ausencia de errores de transcripción**: Se elimina la copia manual de valores.
- **Fácil actualización**: Si los datos cambian, regenera el `.tex` en segundos.

2.6. Nota de arquitectura: independencia y reutilización

Diseño clave: `latex_tools.py` es deliberadamente **agnóstico respecto a la fuente de datos**.

Cualquier módulo que calcule magnitudes numéricas puede usar `latex_tools` para su presentación:

- **Desde** `estadistica.py`: Medias, intervalos de confianza, resultados de tests.
- **Desde** `ajustes.py`: Parámetros ajustados, matrices de covarianza.
- **Desde** `incertidumbres.py`: Magnitudes con Tipo A + Tipo B combinadas.
- **Desde** `montecarlo.py`: Valores medios y desviaciones de simulaciones.

- **Desde** `fft_tools.py`: Amplitudes y frecuencias con incertidumbres.

Esta arquitectura modular permite que cada módulo futuro **contribuya resultados** a `latex_tools`, sin que `latex_tools` tenga que conocer detalles internos de cálculo de ninguno.

Conclusión del capítulo

El módulo `latex_tools.py` cierra el ciclo experimental: desde el análisis en Python hasta la presentación en LATEX. Su diseño minimalista y separado del cálculo lo hace robusto, mantenible, y extensible. Es la capa de **presentación** que todos los módulos científicos de la toolbox comparten.