Aplicaciones computacionales

Ondas gravitacionales {

```
"Sharith Pinzón": "2210709",
    "Angie Sandoval": "2210728",
    "Jorge Silva": "2160411",
    "Vanesa Díaz": "2181334"
}
```

Contenido

01. Introducción {

Las ondas gravitacionales son perturbaciones del espacio-tiempo generadas por sistemas binarios compactos. En particular, la coalescencia de dos estrellas de neutrones produce un chirp: una señal cuya frecuencia y amplitud aumentan con el tiempo hasta la fusión. En este caso, este tramo creciente de la señal es de gran interés, pues codifica detalles sobre la dinámica orbital y la estructura de los objetos involucrados.

¿Qué se hizo? A grandes rasgos....

- Con simulaciones del catálogo SXS se seleccionó el modo dominante (2,2) y su fase estrictamente creciente. Primero se aplicaron distintos métodos de interpolación (Lagrange, Splines y PCHIP) para reconstruir la señal. Después, los datos se dividieron en entrenamiento y prueba, lo que permitió evaluar la precisión de cada método a través de errores cuadráticos y absolutos.

}

02. Objetivo {

Analizar la señal gravitacional de la simulación SXS:NSNS:0001, enfocándonos en la fase del modo dominante (2,2). Para ello, se selecciona el tramo estrictamente creciente y se aplican distintos métodos de interpolación (Lagrange, splines cúbicos y PCHIP). Posteriormente, se construyen grupos de entrenamiento y prueba para evaluar los errores (MSE y MAE) y comparar cuál técnica describe con mayor precisión la dinámica orbital y el chirp de la señal.

}

03. Descripción del sistema físico {

La simulación analizada es SXS:NSNS:0001, que corresponde a la coalescencia de un sistema binario de estrellas de neutrones. Estos objetos ultracompactos, con masas comparables y espín despreciable, se modelan como fuentes de ondas gravitacionales dominadas por su dinámica orbital.

En el código se carga la función de onda completa h y, a partir de ella, se extrae el modo multipolar dominante (ℓ ,m)=(2,2). Sobre este modo se calcula y desenrolla la fase, la cual muestra el chirp, un aumento progresivo de la frecuencia y la amplitud de la señal hasta la fusión.

```
!pip install sxs -q
```

```
import sxs # import sxs
import matplotlib.pyplot as plt
import numpy as np
%config InlineBackend.figure_format = 'retina'
```

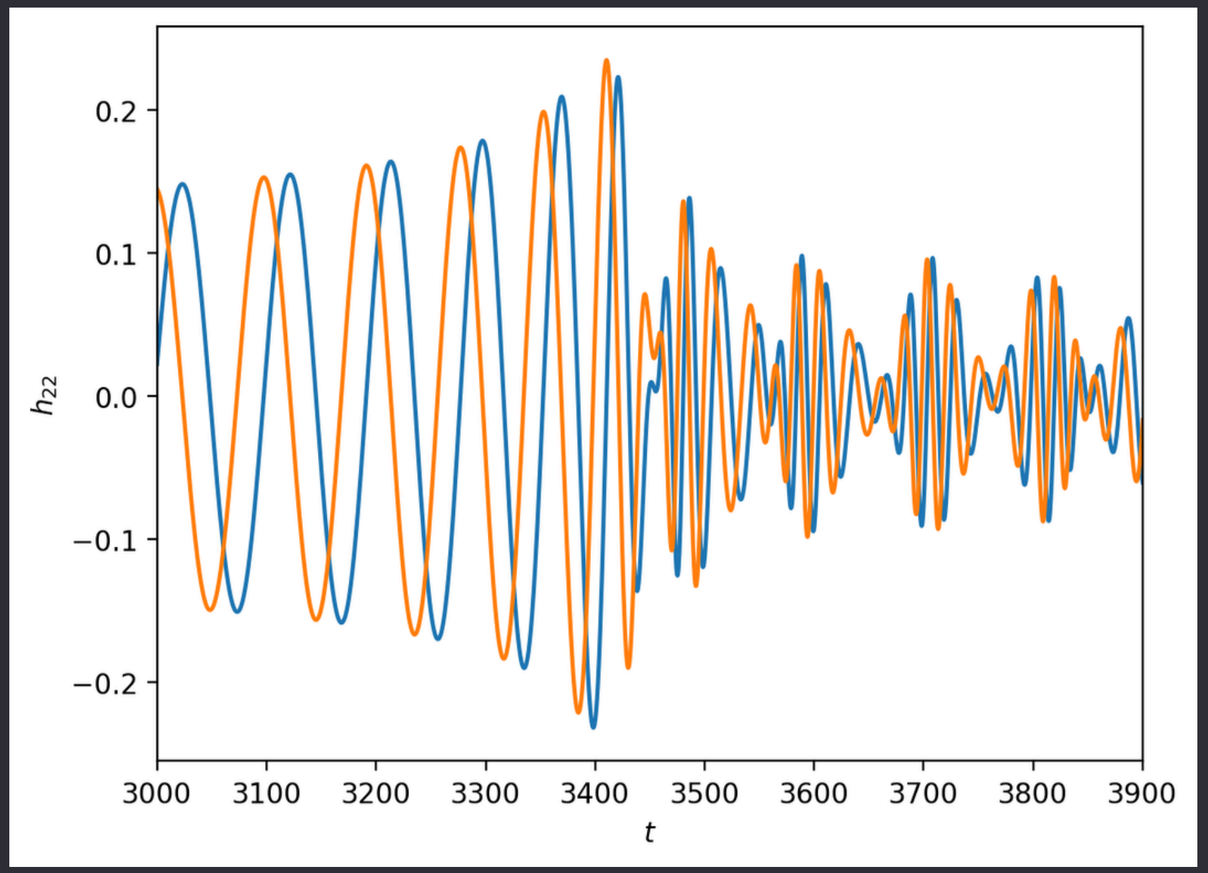
```
sxs_nsns_0001 = sxs.load("SXS:NSNS:0001")
w = sxs_nsns_0001.h
```

```
t0 = w.metadata.reference_time # tiempo de referencia
print(f't0 = {t0}')
```

```
t0 = 392
```

}

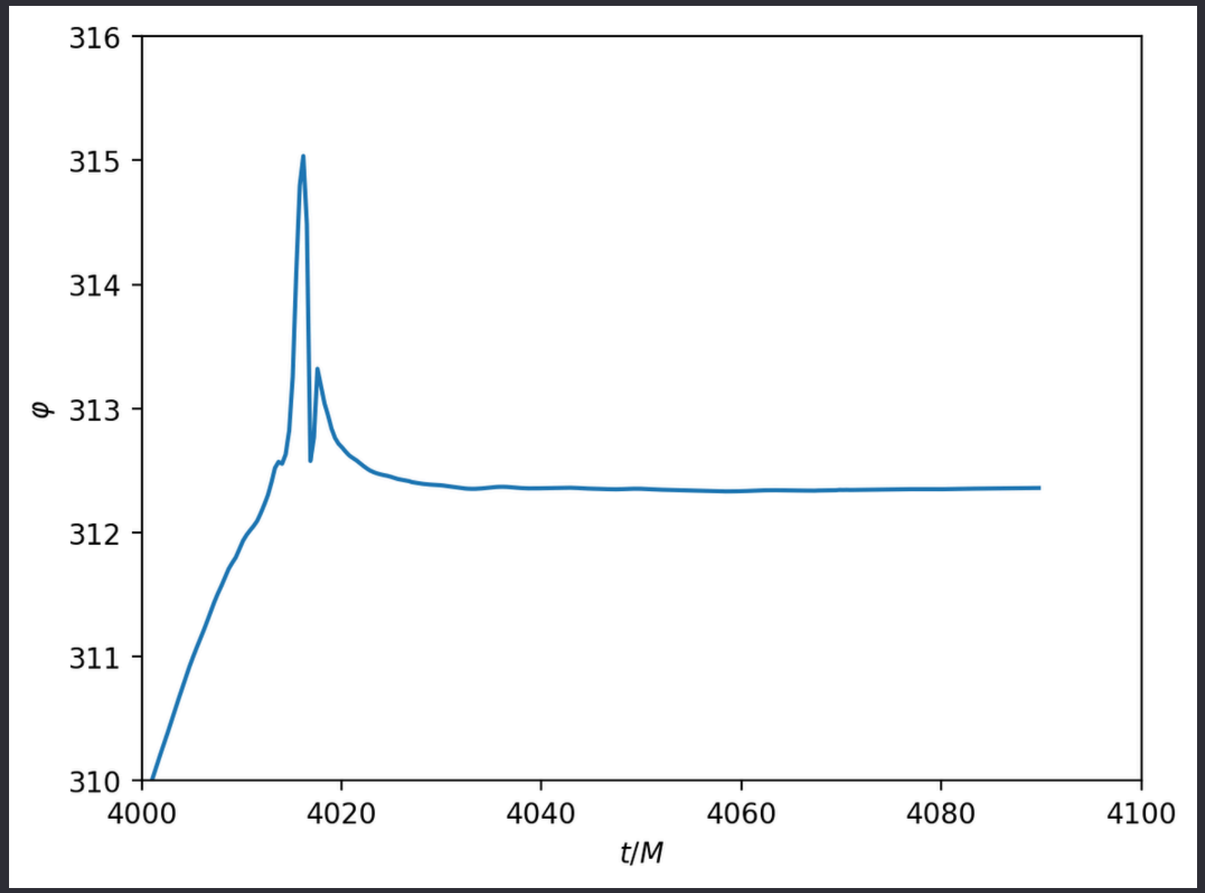
03. Descripción del sistema físico {



La gráfica muestra el modo dominante h_{22} de la onda gravitacional en la simulación SXS:NSNS:0001. Las componentes real (azul) e imaginaria (naranja) crecen en amplitud durante la inspiral, alcanzan un máximo en la fusión de las estrellas de neutrones y luego decaen en la etapa de post-fusión, reflejando la pérdida de energía gravitacional.

}

03. Descripción del sistema físico {

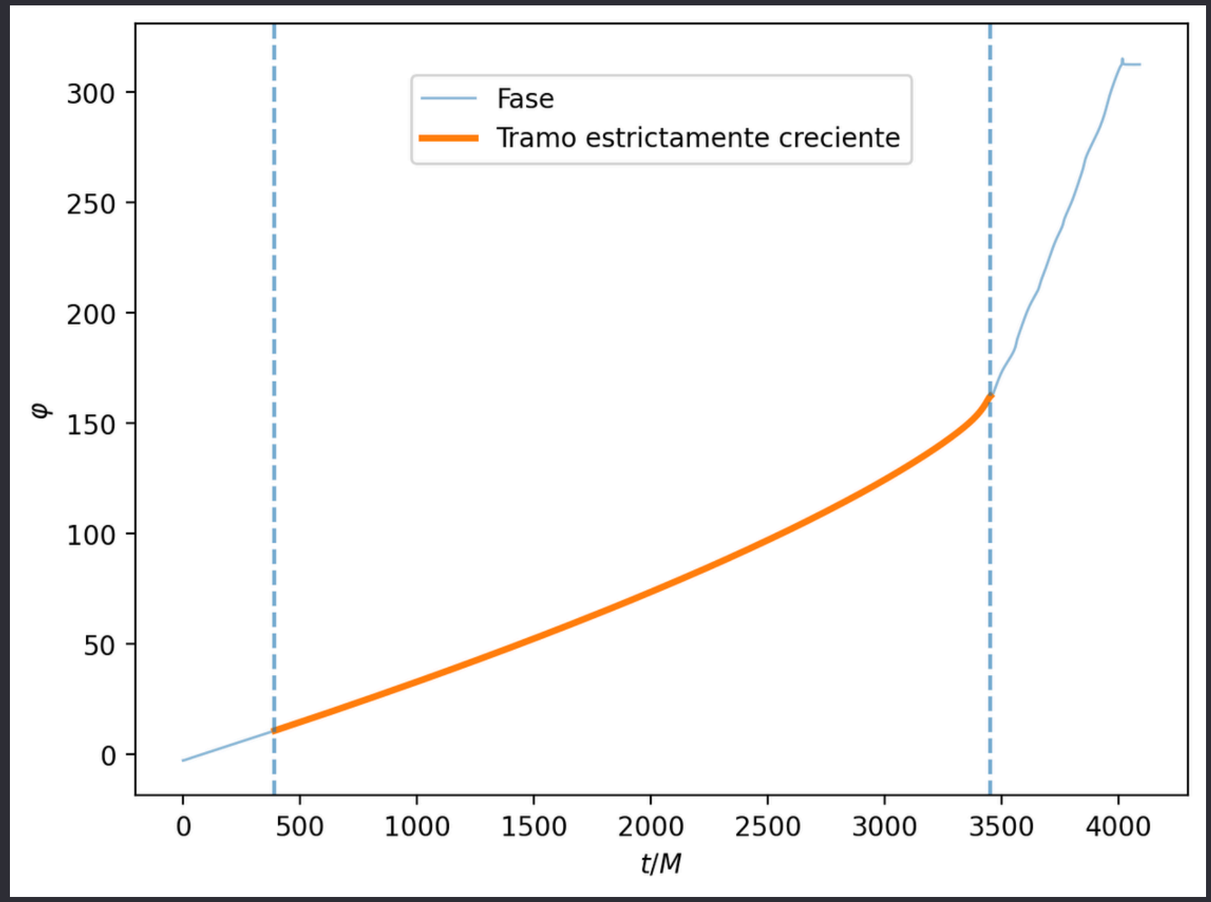


A partir del modo dominante h_{22} se obtiene la fase φ, que refleja la evolución temporal de la señal. Durante la inspiral aumenta de forma continua, alcanza un pico abrupto en la fusión y luego se estabiliza en la post-fusión, mostrando la transición del sistema hacia su relajación.

}

03. Descripción de cómo se seleccionan los datos {

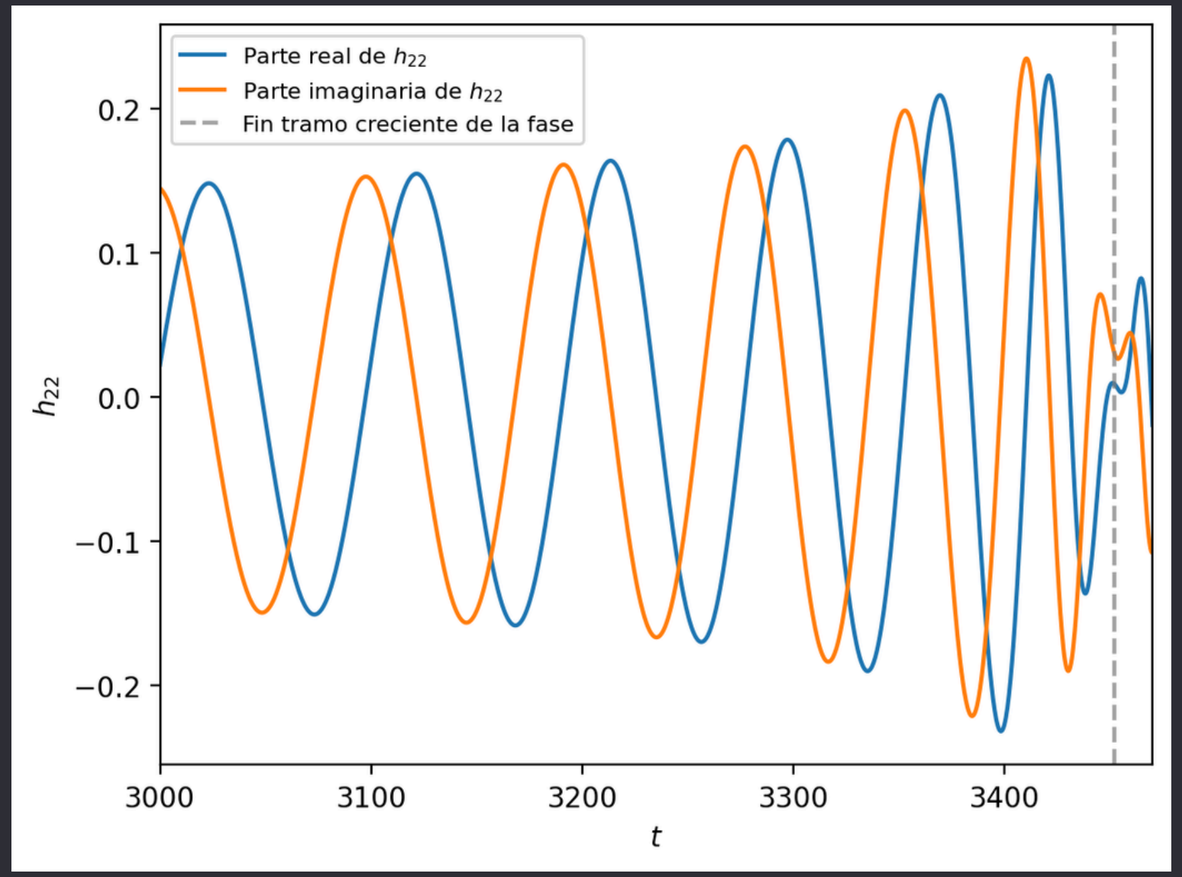
```
t = np.asarray(w_2_2.t)
y = np.asarray(np.unwrap(-np.angle(w_2_2)))

t0  = w.metadata.reference_time
tol = 0.0

i0 = int(np.searchsorted(t, t0, side="left"))
if i0 >= t.size - 1:
    raise ValueError("No hay suficientes puntos a partir de t0.")

dy   = np.diff(y[i0:])
hit  = np.where(dy <= tol)[0]
i1   = i0 + (int(hit[0]) if hit.size else dy.size)

t_inc = t[i0:i1+1]
y_inc = y[i0:i1+1]

t_lin = np.linspace(t_inc[0], t_inc[-1], 600)
y_lin = np.interp(t_lin, t_inc, y_inc)

plt.figure()
plt.plot(t, y, lw=1, alpha=0.5, label="Fase")
plt.plot(t_lin, y_lin, lw=2.5, label="Tramo estrictamente creciente")
plt.axvline(t0, ls="--", alpha=0.6)
plt.axvline(t_inc[-1], ls="--", alpha=0.6)
plt.xlabel(r"$t/M$")
plt.ylabel(r"$\varphi$")
plt.legend(loc="upper center", bbox_to_anchor=(0.5, 0.95))
plt.tight_layout()
plt.show()
```



}

03. Descripción de cómo se seleccionan los datos {

```
plt.figure()
plt.plot(w.t, np.real(w_2_2))
plt.plot(w.t, np.imag(w_2_2))
plt.xlim(3000,3470)
plt.xlabel(r'$t$')
plt.ylabel(r'$h_{22}$')
plt.axvline(t[i1], ls='--', alpha=0.7)
plt.show()
```



}

03. Descripción de cómo se seleccionan los datos {

```
t = np.asarray(w_2_2.t)
y = np.asarray(np.unwrap(-np.angle(w_2_2)))  # fase desenrollada

t0 = w.metadata.reference_time
i0 = int(np.searchsorted(t, t0, side="left"))

dy  = np.diff(y[i0:])
hit = np.where(dy <= 0.0)[0]
i1  = i0 + (int(hit[0]) if hit.size else dy.size)

t_inc = t[i0:i1+1]
y_inc = y[i0:i1+1]
```

A) ¿Cuántos datos hay?

```
t = np.asarray(w_2_2.t)
y = np.asarray(np.unwrap(-np.angle(w_2_2)))

t0  = w.metadata.reference_time
i0  = int(np.searchsorted(t, t0, side="left"))

dy  = np.diff(y[i0:])
hit = np.where(dy <= 0.0)[0]
i1  = i0 + (int(hit[0]) if hit.size else dy.size)

t_inc = t[i0:i1+1]
y_inc = y[i0:i1+1]

n_total = t.size
n_tramo = t_inc.size
print(f"Total puntos (2,2): {n_total}")
print(f"Puntos en tramo estrictamente creciente: {n_tramo}")
```

```
Total puntos (2,2): 11452
Puntos en tramo estrictamente creciente: 8568
```

}

03. Descripción de cómo se seleccionan los datos {

B) Máximo y mínimos

```
i_min = np.argmin(y_inc)
i_max = np.argmax(y_inc)

t_min, phi_min = t_inc[i_min], y_inc[i_min]
t_max, phi_max = t_inc[i_max], y_inc[i_max]

print(f"mínimo: ϕ={phi_min:.9g} en t={t_min:.9g}")
print(f"máximo: ϕ={phi_max:.9g} en t={t_max:.9g}")
```

```
mínimo: ϕ=10.5727411 en t=392.018076
máximo: ϕ=162.081486 en t=3451.94944
```

}

03. Descripción de cómo se seleccionan los datos {

C) ¿El paso temporal es constante?

```
dt_full  = np.diff(t)
dt_tramo = np.diff(t_inc)

def es_constante(dt, rtol=1e-10, atol=1e-12):
    return np.allclose(dt, dt[0], rtol=rtol, atol=atol)

print("Paso constante (full): ", es_constante(dt_full))
print("Paso constante (tramo):", es_constante(dt_tramo))
print(f"dt_tramo promedio = {dt_tramo.mean():.9g}, min = {dt_tramo.min():.9g}, max = {dt_tramo.max():.9g}")
```

```
Paso constante (full):  False
Paso constante (tramo): False
dt_tramo promedio = 0.357176533, min = 0.357131478, max = 0.358895555
```

}

04. Métodos de interpolación {

Interpolación polinómica de Lagrange

Interpolacion CubicSpline

- natural
- not a knot

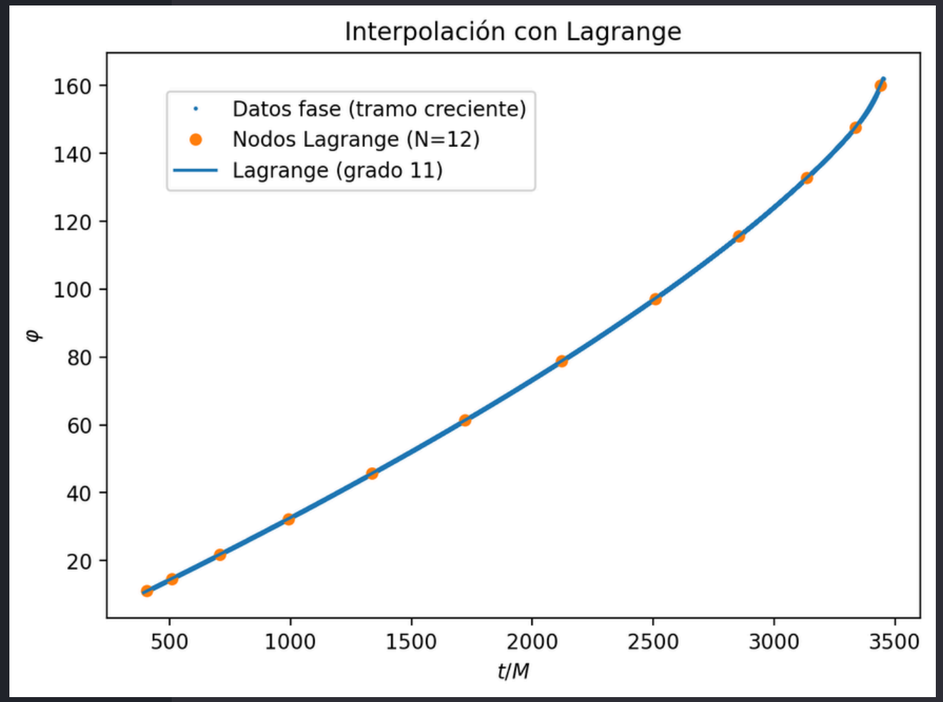
Interpolación PCHIP(Polynomial Cubic Hermite Interpolating Polynomial)

}

Interpolación polinómica de Lagrange {

La interpolación de Lagrange construye un polinomio que pasa exactamente por un conjunto de puntos dados usando polinomios base Li(x) que son 1 en su punto y 0 en los demás. Es directa y no requiere resolver sistemas, pero puede ser inestable para muchos puntos.

```python
from scipy.interpolate import lagrange

a, b = t_inc[0], t_inc[-1]

N = 12
k = np.arange(N)
t_cheb = 0.5*(a+b) + 0.5*(b-a)*np.cos((2*k+1)/(2*N)*np.pi)

idx = np.clip(np.searchsorted(t_inc, t_cheb), 0, t_inc.size-1)
idx = np.unique(idx)  # evita repetidos

P = lagrange(t_inc[idx], y_inc[idx])

t_dense = np.linspace(a, b, 800)
y_dense = P(t_dense)

plt.figure()
plt.plot(t_inc, y_inc, '.', ms=2, color="tab:blue", label="Datos fase (tramo creciente)")
plt.plot(t_inc[idx], y_inc[idx], 'o', ms=5, color="tab:orange", label=f"Nodos Lagrange (N={idx.size})")
plt.plot(t_dense, y_dense, '-', color="tab:blue", lw=1.5, label=f"Lagrange (grado {idx.size-1})")
plt.title("Interpolación con Lagrange")
plt.xlabel(r"$t/M$")
plt.ylabel(r"$\varphi$")
plt.legend(loc="upper center", bbox_to_anchor=(0.3, 0.95))
plt.tight_layout()
plt.show()
```

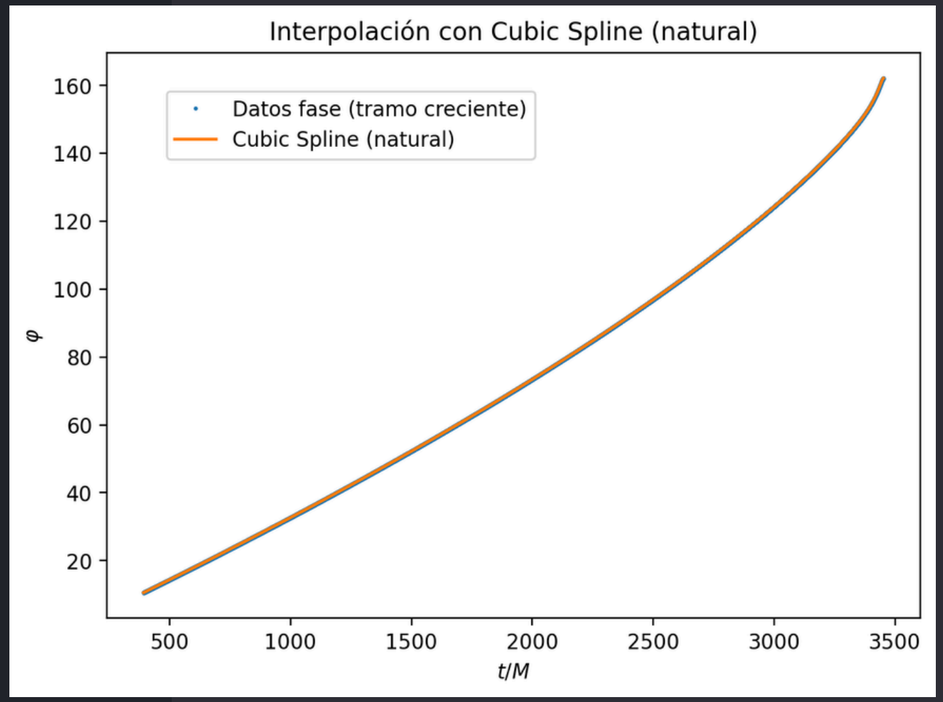


}

Interpolación CubicSpline (natural) {

Los splines cúbicos construyen un polinomio cúbico por tramos entre cada par de nodos, asegurando continuidad de primer y segundo derivada. **Condición natural**: la segunda derivada en los extremos es cero, y’’(a) = y’’(b) = 0, lo que suaviza la interpolación en los bordes.

```python
from scipy.interpolate import CubicSpline

spline = CubicSpline(t_inc, y_inc, bc_type='natural')

t_dense = np.linspace(t_inc[0], t_inc[-1], 600)

y_spline = spline(t_dense)

plt.figure()
plt.plot(t_inc, y_inc, '.', ms=2, color="tab:blue", label="Datos fase (tramo creciente)")
plt.plot(t_dense, y_spline, '-', color="tab:orange", lw=1.5, label="Cubic Spline (natural)")
plt.title("Interpolación con Cubic Spline (natural)")
plt.xlabel(r"$t/M$")
plt.ylabel(r"$\varphi$")
plt.legend(loc="upper center", bbox_to_anchor=(0.3, 0.95))
plt.tight_layout()
plt.show()
```

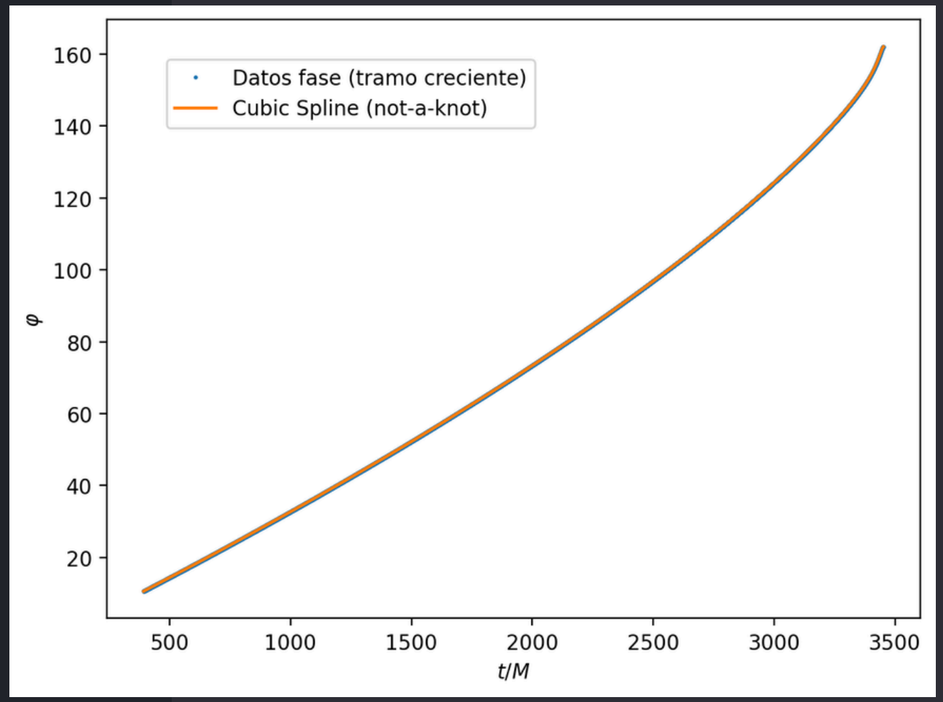


}

Interpolación CubicSpline (not a knot) {

Este método es una variante de spline cúbico. **Condición not-a-knot**: el tercer derivado es continuo en los dos nodos adyacentes a cada extremo. Esto produce una interpolación más suave y evita restricciones artificiales en los extremos.

```python
from scipy.interpolate import CubicSpline

cs_nak = CubicSpline(t_inc, y_inc, bc_type='not-a-knot')

t_dense = np.linspace(t_inc[0], t_inc[-1], 600)
y_nak   = cs_nak(t_dense)

plt.figure()
plt.plot(t_inc, y_inc, '.', ms=2, color="tab:blue",   label="Datos fase (tramo creciente)")
plt.plot(t_dense, y_nak, '-', ms=3, color="tab:orange", label="Cubic Spline (not-a-knot)")
plt.xlabel(r"$t/M$")
plt.ylabel(r"$\varphi$")
plt.legend(loc="upper center", bbox_to_anchor=(0.3, 0.95))
plt.tight_layout()
plt.show()
```

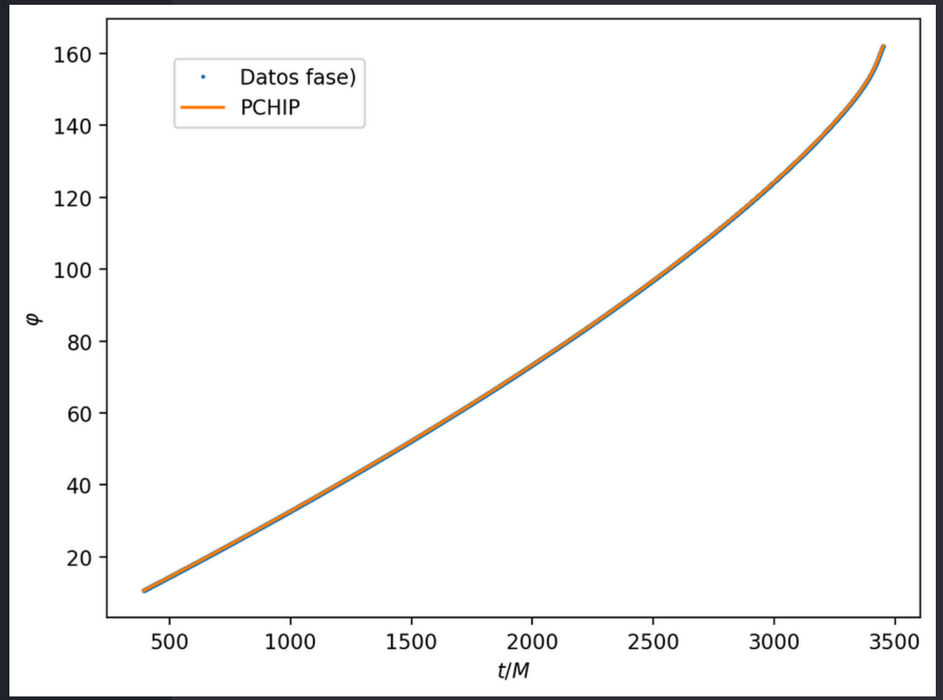


}

Interpolación PCHIP(Piecewise Cubic Hermite Interpolating Polynomial) {

Construye un spline cúbico por tramos que preserva la forma (shape-preserving). Si los datos son crecientes o decrecientes, la interpolación no introduce oscilaciones artificiales.

```python
from scipy.interpolate import PchipInterpolator

pchip = PchipInterpolator(t_inc, y_inc)

t_dense = np.linspace(t_inc[0], t_inc[-1], 600)
y_pchip  = pchip(t_dense)

plt.figure()
plt.plot(t_inc, y_inc, '.', ms=2, color="tab:blue",   label="Datos fase)")
plt.plot(t_dense, y_pchip, '-', ms=3, color="tab:orange", label="PCHIP")
plt.xlabel(r"$t/M$")
plt.ylabel(r"$\varphi$")
plt.legend(loc="upper center", bbox_to_anchor=(0.2, 0.95))
plt.tight_layout()
plt.show()
```



}

05. Grupos de prueba y entrenamiento {

```python
n_total = 80
idx_all = np.arange(len(t_inc))
sel_idx = np.linspace(0, len(idx_all) - 1, n_total, dtype=int)

train_idx = sel_idx[::3]
train_idx = np.union1d(train_idx, sel_idx[1::3])
test_idx  = np.setdiff1d(sel_idx, train_idx)

train_idx = np.union1d(train_idx, [len(idx_all) - 1])

t_train_lagr, y_train_lagr = t_inc[train_idx], y_inc[train_idx]
t_test_lagr,  y_test_lagr  = t_inc[test_idx],  y_inc[test_idx]

print(f"[Lagrange] Entrenamiento: {t_train_lagr.size} | Prueba: {t_test_lagr.size} | Total muestreados: {sel_idx.size}")

from scipy.interpolate import lagrange

a_lagr, b_lagr = t_train_lagr.min(), t_train_lagr.max()
N_nodes = 12
k = np.arange(N_nodes)
t_cheb_lagr = 0.5*(a_lagr+b_lagr) + 0.5*(b_lagr-a_lagr)*np.cos((2*k+1)/(2*N_nodes)*np.pi)

idx_nodes_lagr = np.clip(np.searchsorted(t_train_lagr, t_cheb_lagr), 0, t_train_lagr.size-1)
idx_nodes_lagr = np.unique(idx_nodes_lagr)

P_lagr = lagrange(t_train_lagr[idx_nodes_lagr], y_train_lagr[idx_nodes_lagr])

t_dense_lagr = np.linspace(t_inc[0], t_inc[-1], 800)
phi_lagr_dense = P_lagr(t_dense_lagr)
```

Se generan 80 índices equiespaciados del rango completo (sel_idx), tomados de t_inc por posición.

Se arma el entrenamiento tomando dos de cada tres de esos índices: sel_idx[::3] (1º de cada terna) ∪ sel_idx[1::3] (2º de cada terna) → ≈ 2/3 del total.

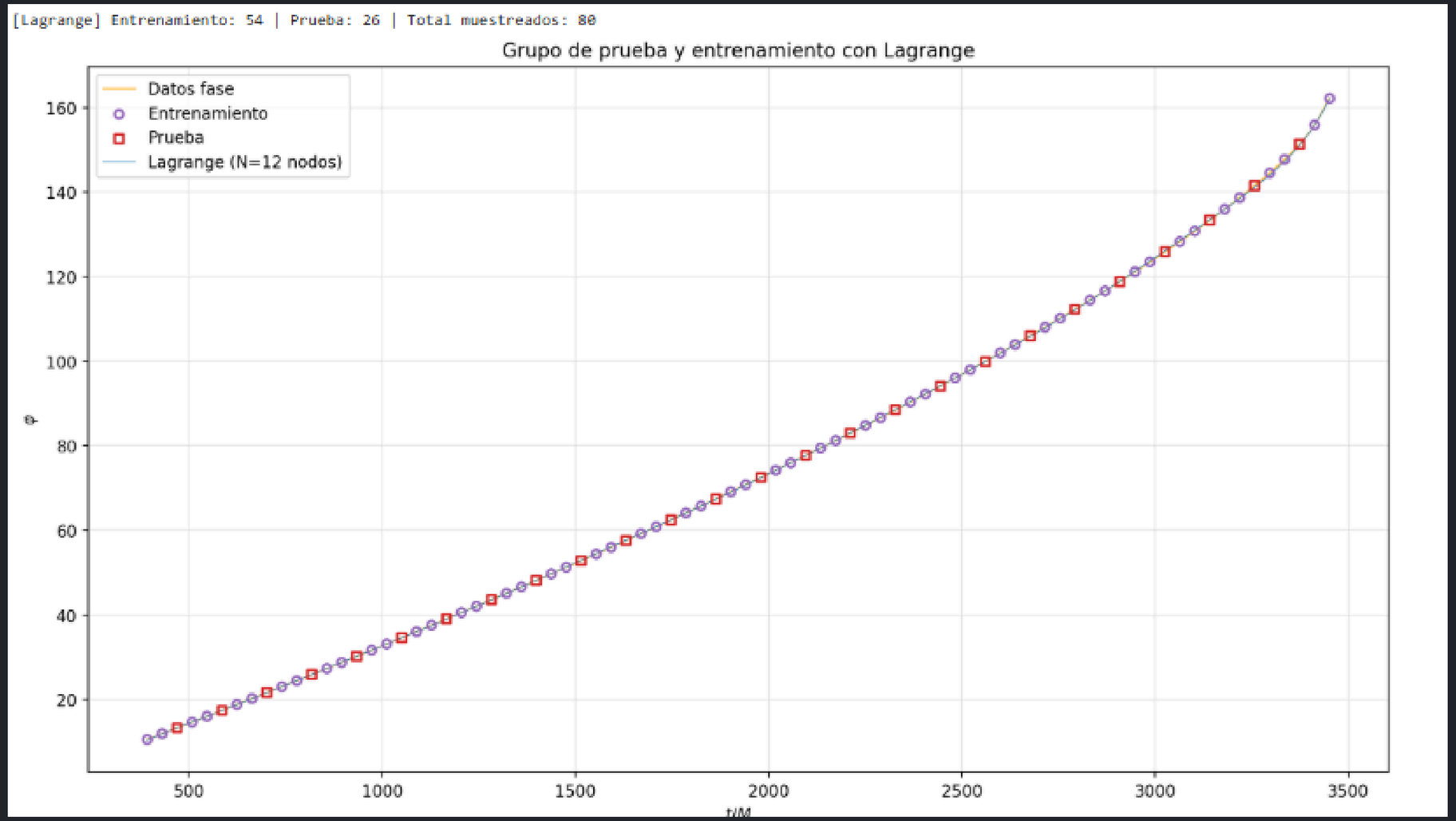
El conjunto de prueba son los restantes (básicamente sel_idx[2::3]) obtenidos como diferencia: setdiff1d(sel_idx, train_idx) → ≈ 1/3.

Se fuerza incluir el último dato global en entrenamiento (len(idx_all)-1) para evitar extrapolación al borde.

Finalmente, esos índices se aplican sobre t_inc y y_inc para obtener (t_train, y_train) y (t_test, y_test).

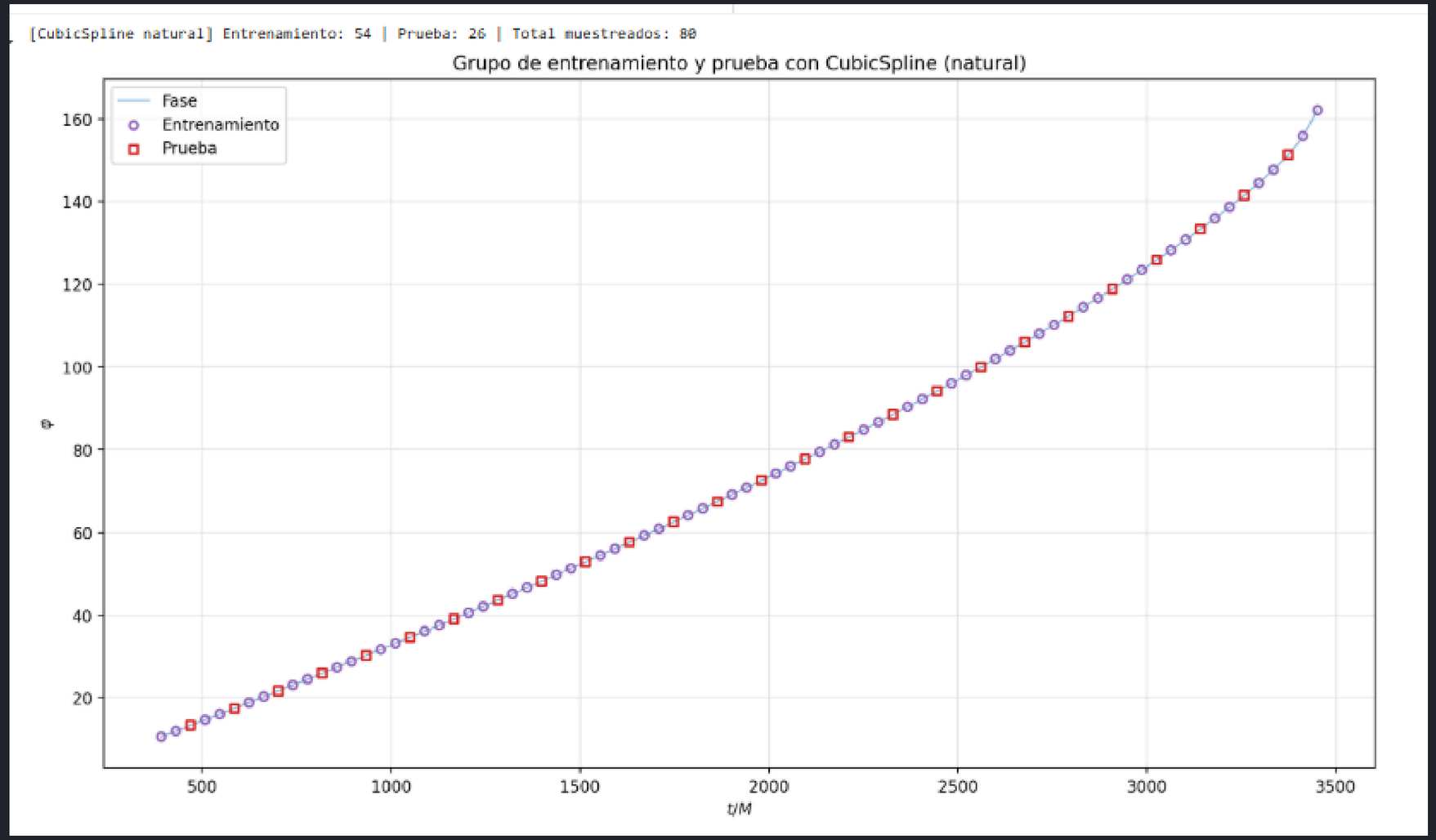
}

Grupo de entrenamiento y de prueba para Interpolación Lagrange {



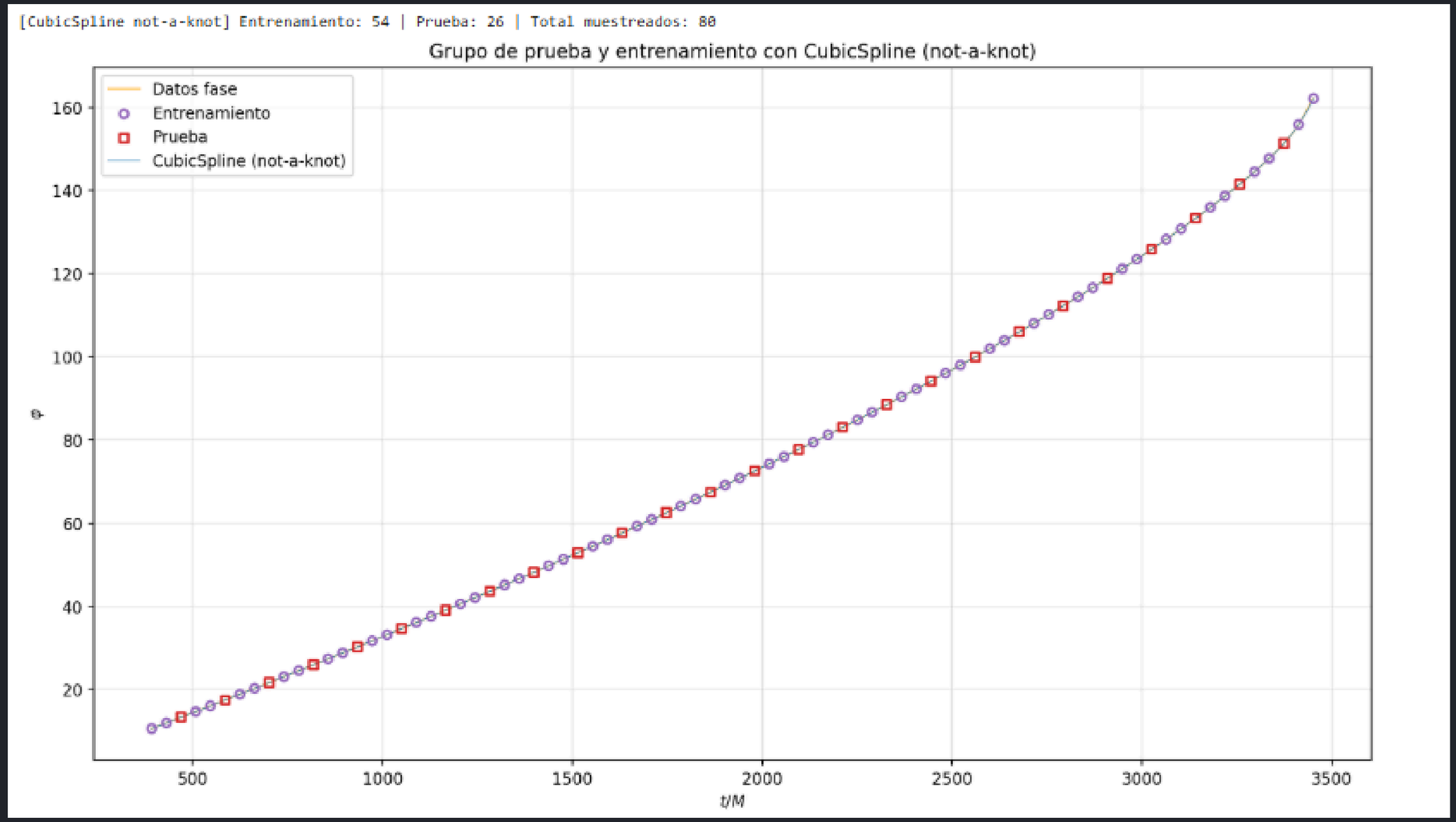
}

Grupo de entrenamiento y de prueba para Interpolación CubicSpline (natural) {



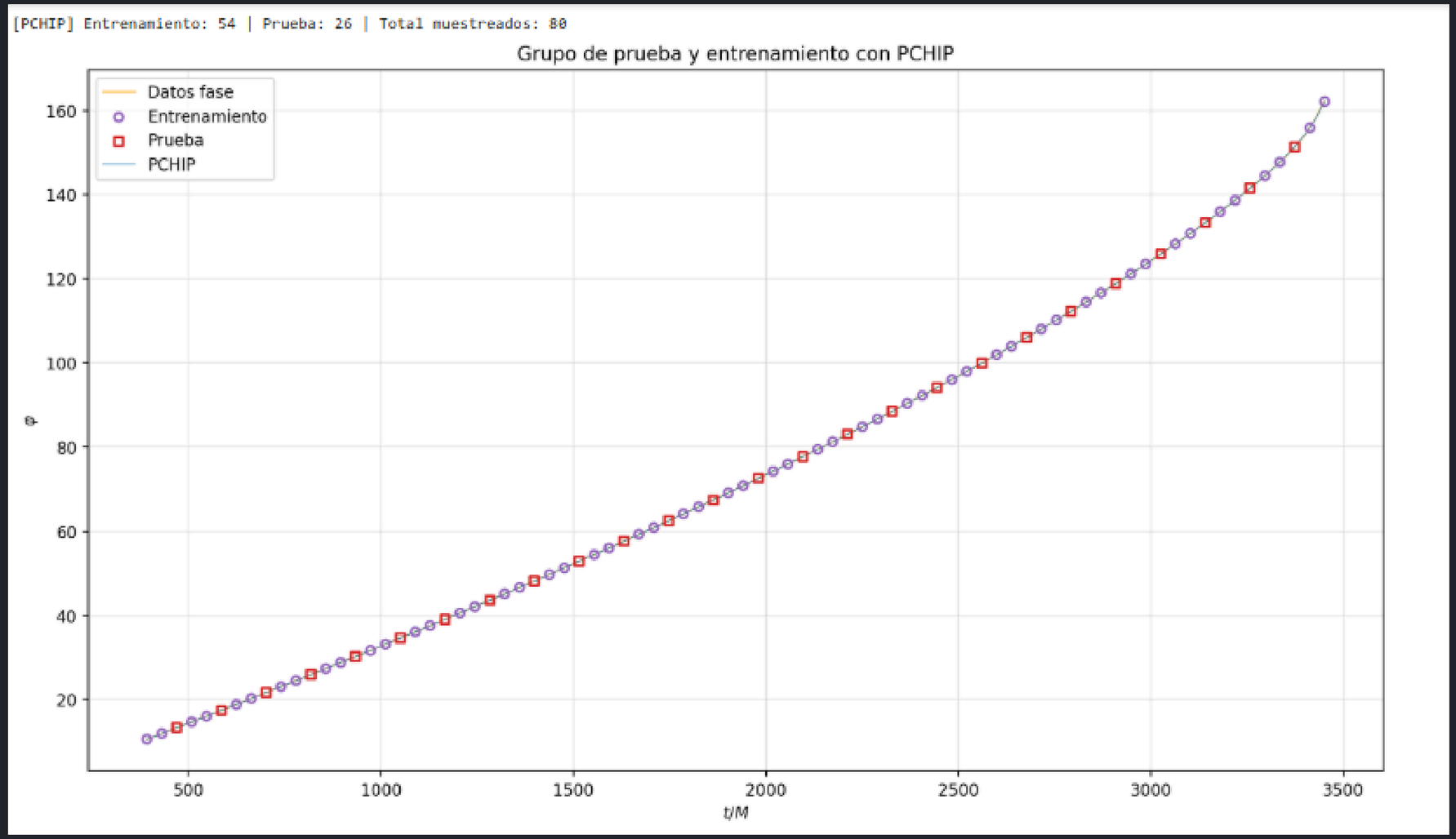
}

Grupo de entrenamiento y de prueba para Interpolación CubicSpline (not a knot) {



}

Grupo de entrenamiento y de prueba para Interpolación PCHIP {



}

06. Error con grupo de prueba y entrenamiento {

Interpolación polinómica de Lagrange

Interpolacion CubicSpline

- natural
- not a knot

Interpolación PCHIP(Polynomial Cubic Hermite Interpolating Polynomial)

}

06. Error para Lagrange {

```python
# Predicciones con Lagrange
# ==========================
y_hat_train_lagr = P_lagr(t_train_lagr)
y_hat_test_lagr  = P_lagr(t_test_lagr)

# ==========================
# Errores punto a punto
# ==========================
SE_train_lagr = (y_train_lagr - y_hat_train_lagr)**2
SE_test_lagr  = (y_test_lagr - y_hat_test_lagr)**2

# Error absoluto
AE_train_lagr = np.abs(y_train_lagr - y_hat_train_lagr)
AE_test_lagr  = np.abs(y_test_lagr - y_hat_test_lagr)

# ==========================
# Errores promedio
# ==========================
MSE_train_lagr = SE_train_lagr.mean()
MSE_test_lagr  = SE_test_lagr.mean()

MAE_train_lagr = AE_train_lagr.mean()
MAE_test_lagr  = AE_test_lagr.mean()

print("===== Resultados globales Lagrange =====")
print(f"Entrenamiento -> MSE = {MSE_train_lagr:.6e}, MAE = {MAE_train_lagr:.6e}")
print(f"Prueba        -> MSE = {MSE_test_lagr:.6e}, MAE = {MAE_test_lagr:.6e}")
```

Usa el polinomio de Lagrange P_lagr para predecir los valores en los conjuntos de entrenamiento y prueba (y_hat_train_lagr, y_hat_test_lagr).

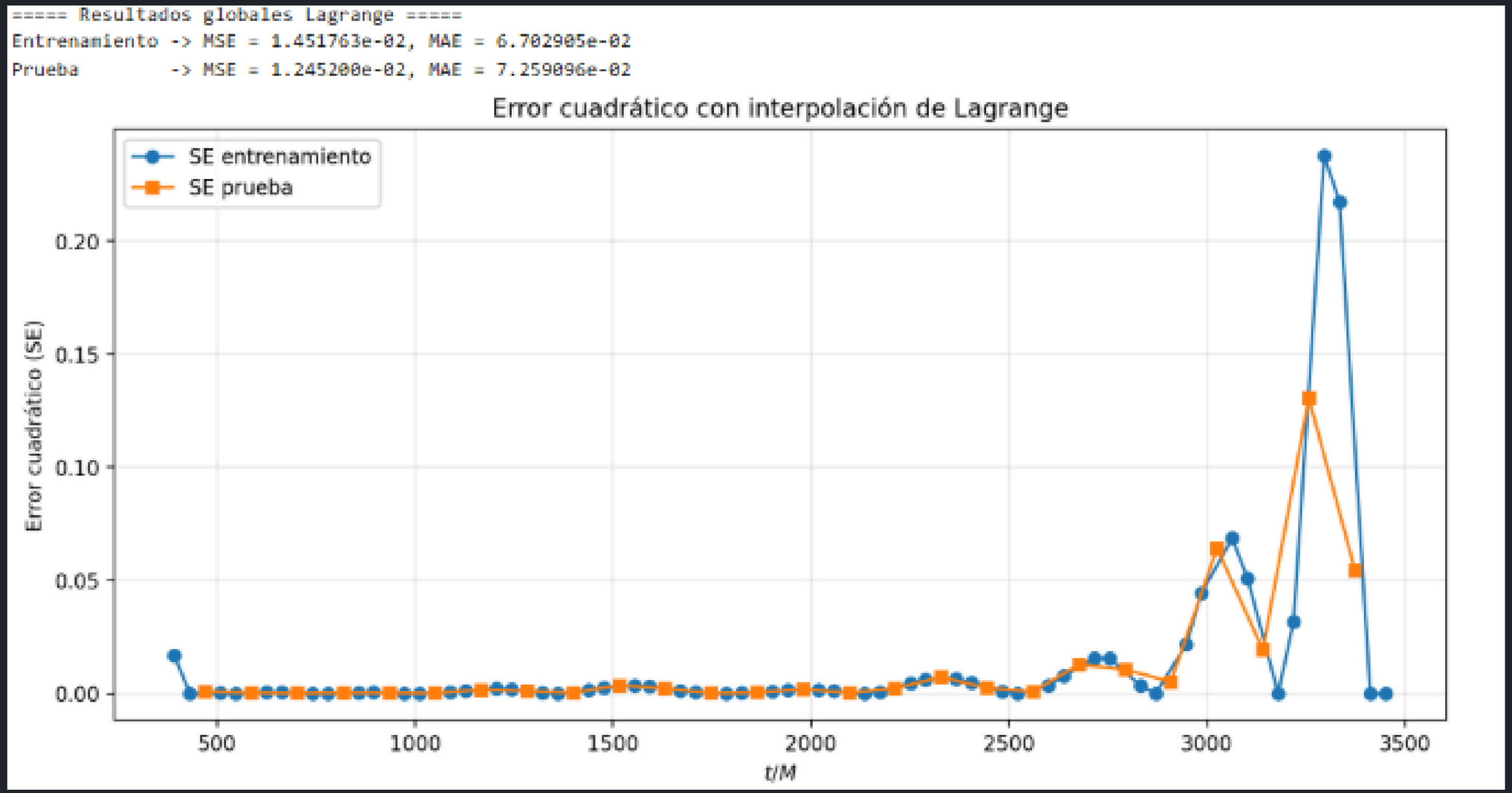
Calcula los errores punto a punto: cuadrático (SE) y absoluto (AE) comparando datos reales vs. predichos en cada conjunto.

Obtiene las métricas promedio: MSE (promedio de SE) y MAE (promedio de AE) para entrenamiento y prueba.

}

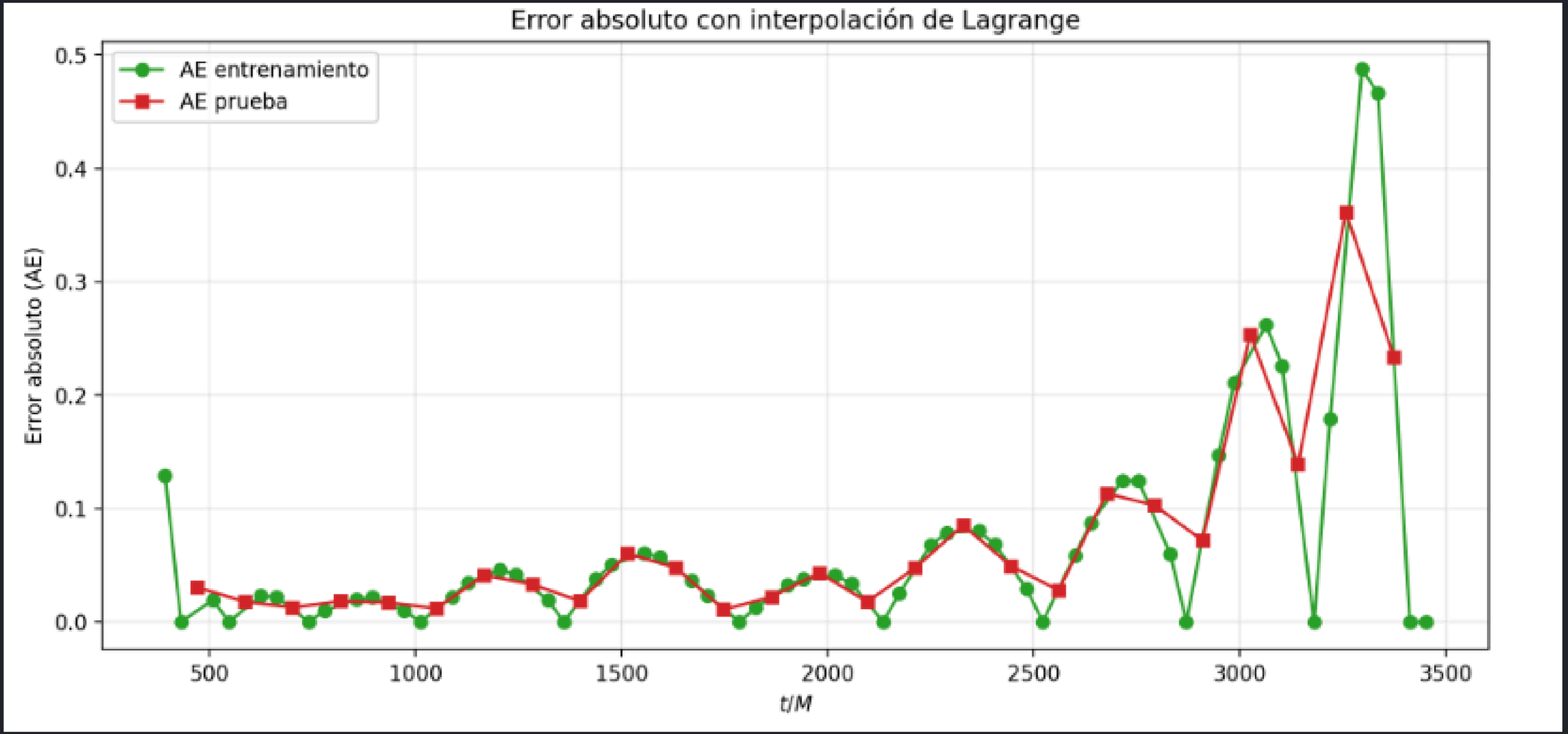
Error cuadrático para Lagrange {

```
===== Resultados globales Lagrange =====
Entrenamiento -> MSE = 1.451763e-02, MAE = 6.702905e-02
Prueba        -> MSE = 1.245200e-02, MAE = 7.259096e-02
```



}

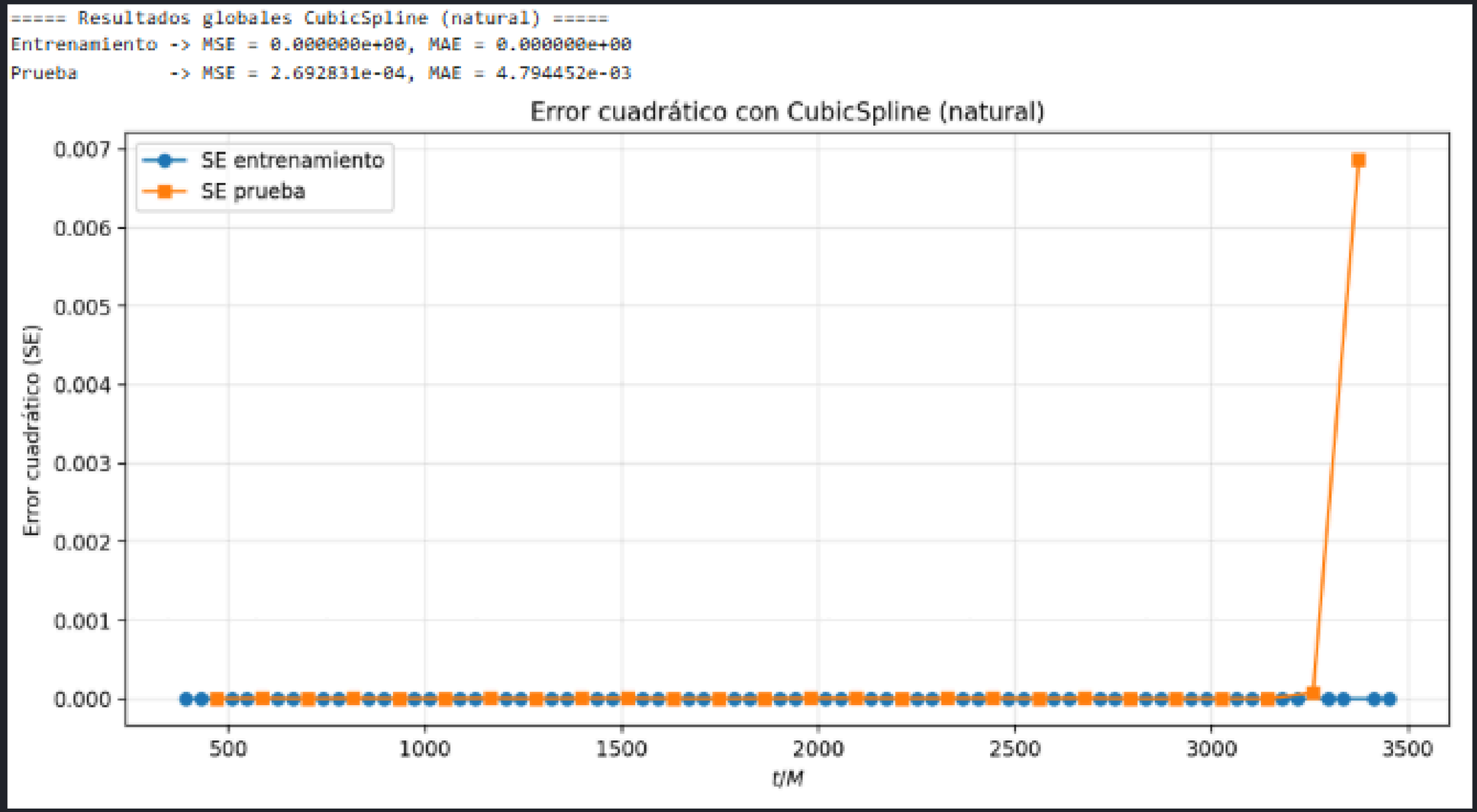
Error absoluto para Lagrange {



}

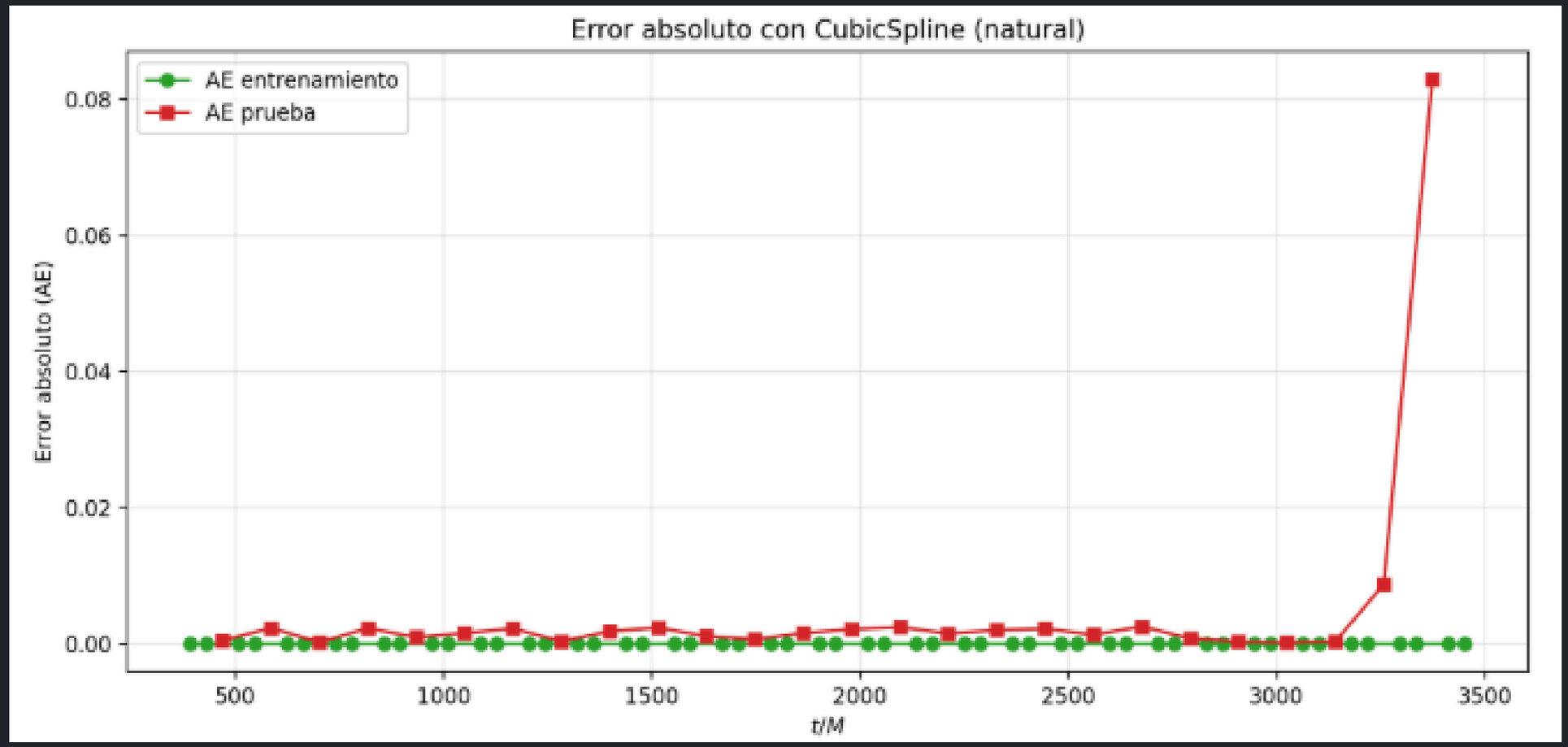
Error cuadrático para CubicSpline (natural) {

```
===== Resultados globales CubicSpline (natural) =====
Entrenamiento -> MSE = 0.000000e+00, MAE = 0.000000e+00
Prueba        -> MSE = 2.692831e-04, MAE = 4.794452e-03
```



}

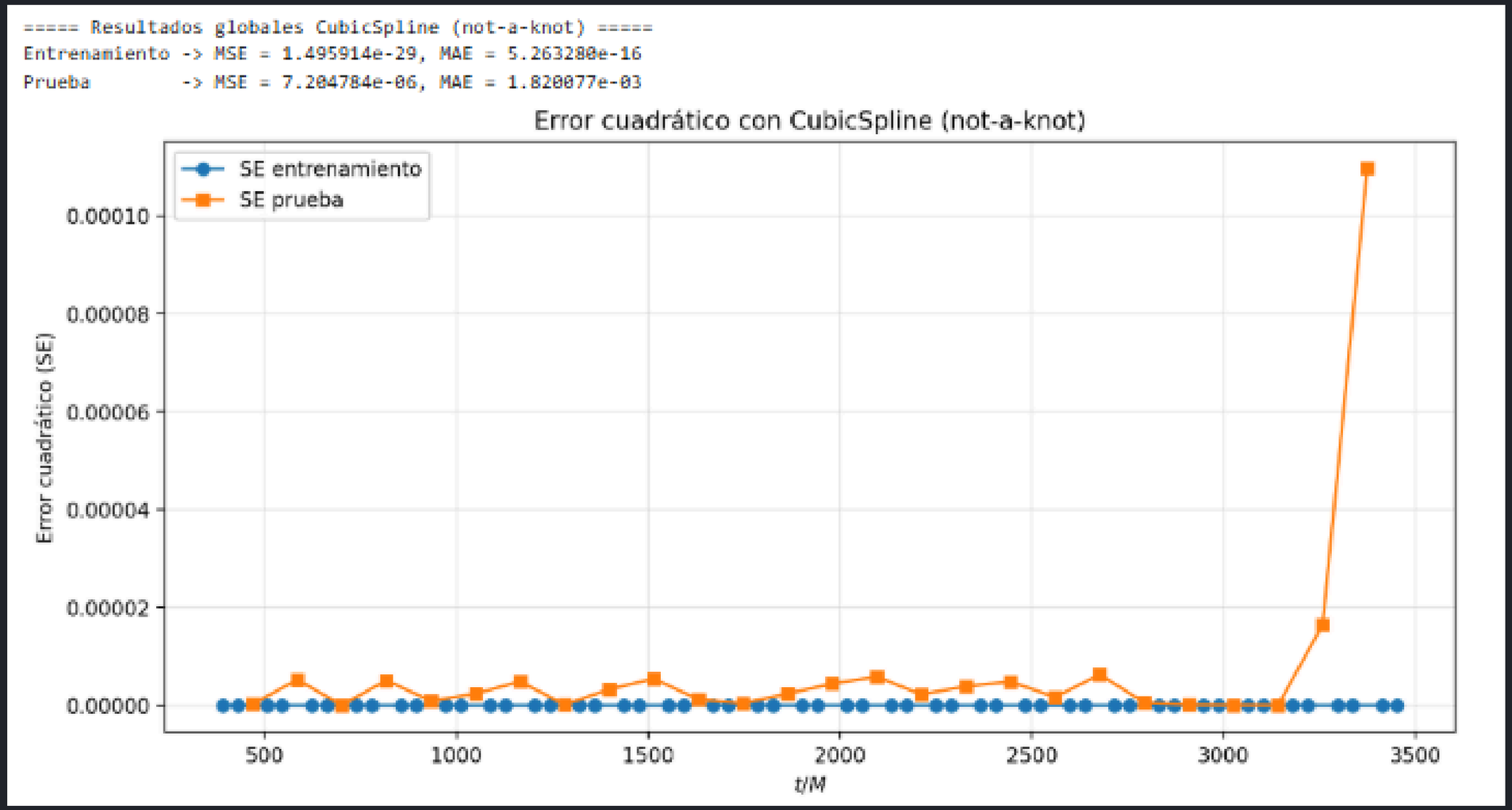
Error absoluto para CubicSpline (natural) {



}

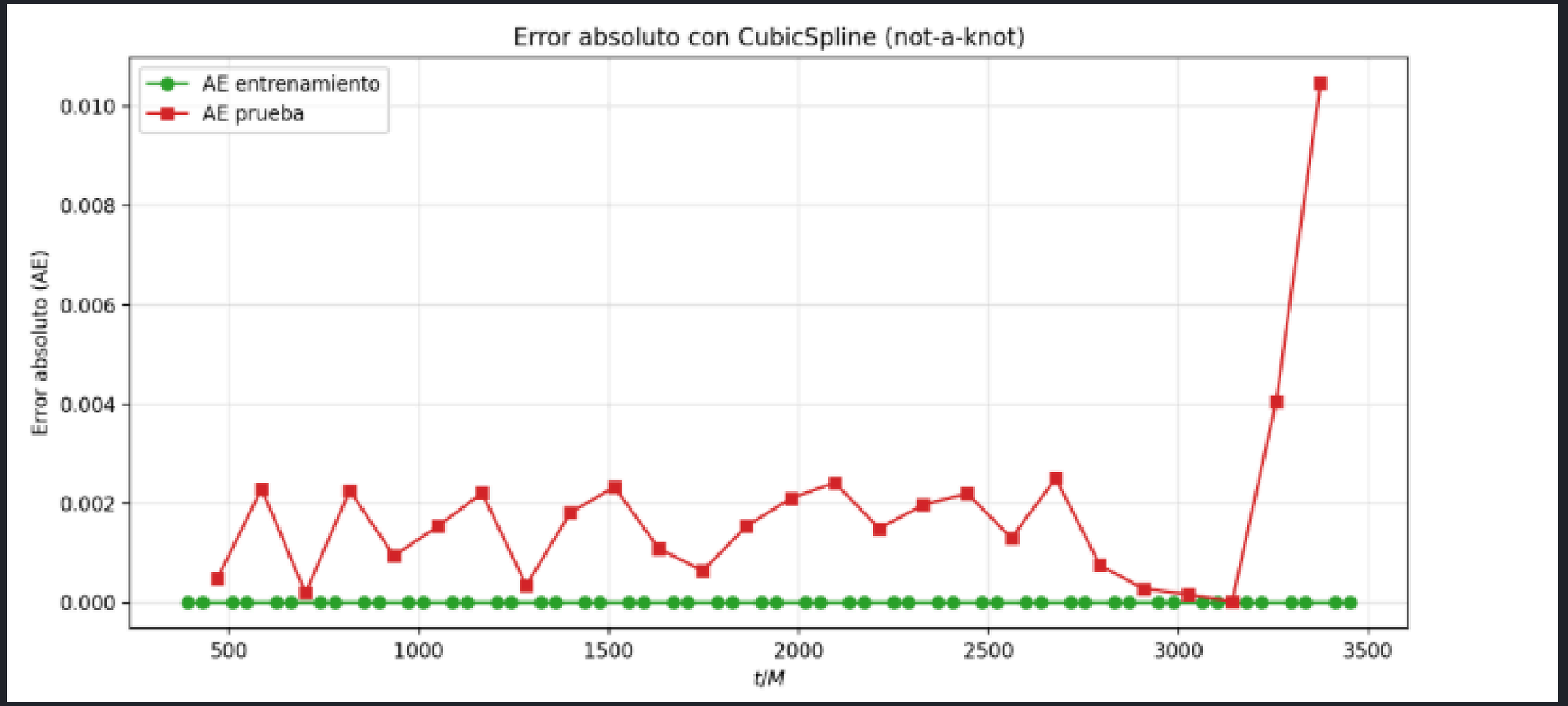
Error cuadrático para CubicSpline (not-a-knot) {

```
===== Resultados globales CubicSpline (not-a-knot) =====
Entrenamiento -> MSE = 1.495914e-29, MAE = 5.263280e-16
Prueba        -> MSE = 7.204784e-06, MAE = 1.820077e-03
```



}

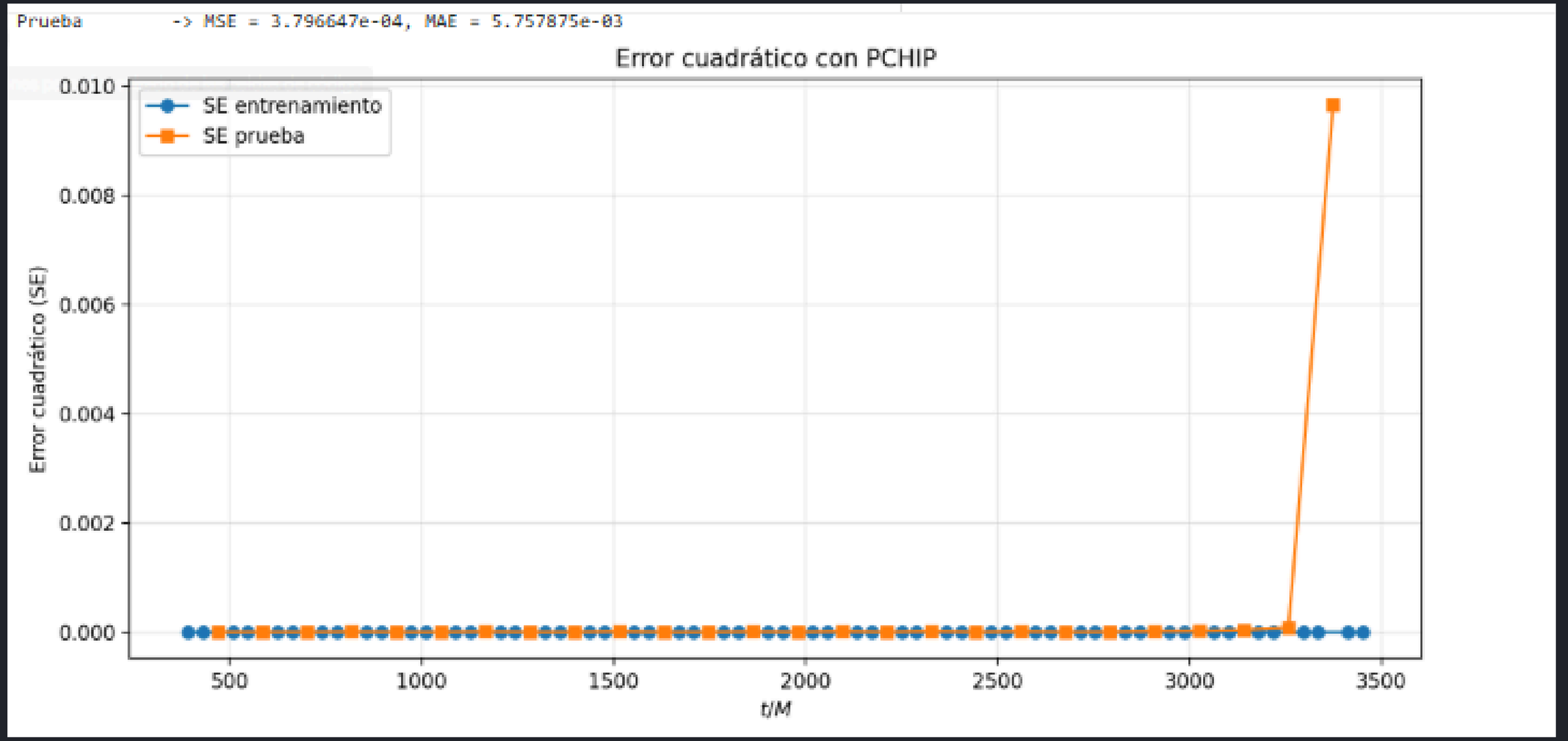
Error absoluto para CubicSpline (not-a-knot) {



}

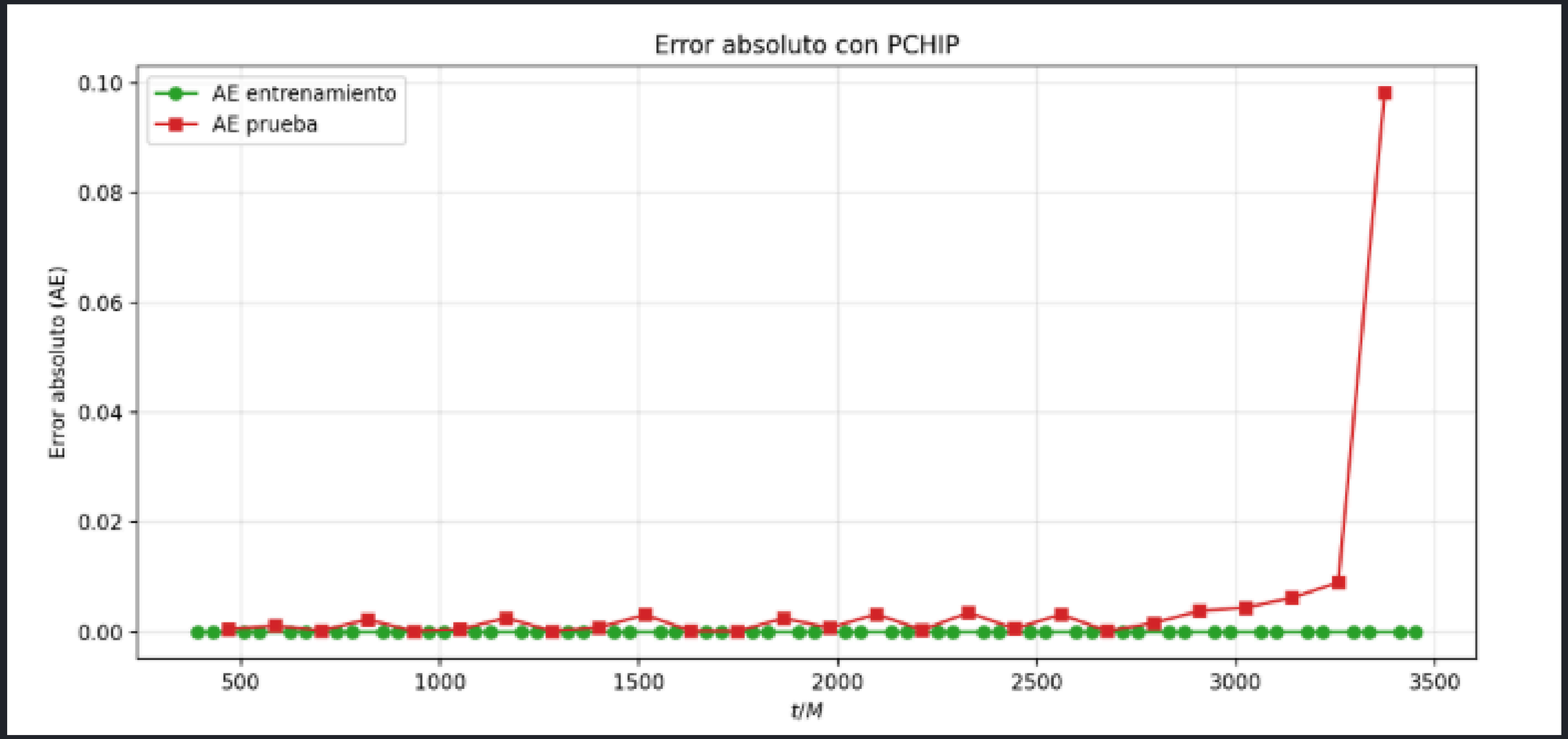
Error cuadrático para PCHIP {

Prueba -> MSE = 3.796647e-04, MAE = 5.757875e-03



}

Error absoluto para PCHIP {



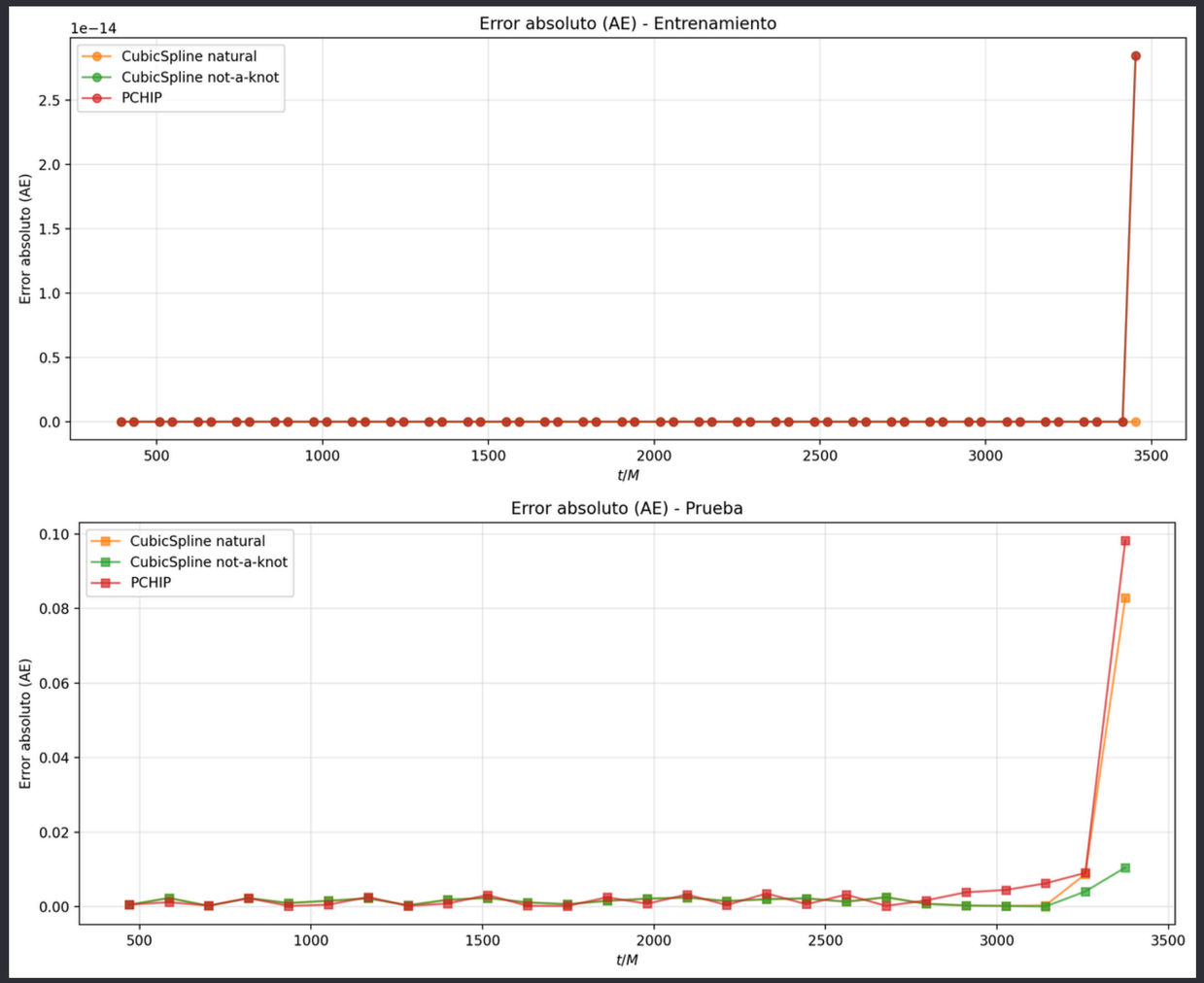
}

¿Qué método es mejor? {

Se llego a esta conclusión a partir del análisis de error absoluto para los grupos de entrenamiento y de prueba

	Método	MAE Entrenamiento	MAE Prueba
0	Lagrange	6.702905e-02	7.259096e-02
1	CubicSpline natural	0.000000e+00	4.794452e-03
2	CubicSpline not-a-knot	5.263280e-16	1.820077e-03
3	PCHIP	5.263280e-16	5.757875e-03

Interpolación CubicSpline (not a knot)



}

Gracias {

```
"Sharith Pinzón": "2210709",
    "Angie Sandoval": "2210728",
    "Jorge Silva": "2160411",
    "Vanesa Díaz": "2181334"
```

}