



**UNIVERSIDAD COMPLUTENSE DE MADRID**

FACULTAD DE CIENCIAS MATEMATICAS

# ANALISIS NUMERICO DE EDP's: PRACTICA 1

*Problema mal puesto - Taylor - Fourier*

Autor:

Jon Mañueco Rubio

Febrero de 2023

## Índice

<b>1. Ejercicio 1</b>	<b>2</b>
<b>2. Ejercicio 2</b>	<b>3</b>
<b>3. Ejercicio 3</b>	<b>10</b>
<b>4. Anexo</b>	<b>15</b>
4.1. Código . . . . .	15

## 1. Ejercicio 1

**Ejercicio 1.** Considere el siguiente problema de evolución:

$$\begin{cases} u_t(x, y, t) = 2u_{xx}(x, y, t) - u_{yy}(x, y, t) & t > 0, (x, y) \in (0, 1)^2 \\ u(x, y, 0) = \sin(\pi x) \sin(\pi y) & (x, y) \in (0, 1)^2 \\ u(x, 0, t) = u(x, 1, t) = u(0, y, t) = u(1, y, t) = 0 & t > 0, (x, y) \in (0, 1)^2 \end{cases} \quad (1)$$

donde  $u(x, y, t)$  está definida en  $(x, y) \in [0, 1]^2$  y  $t \geq 0$ , tiene dos derivadas en  $x$  e  $y$  y una derivada en  $t$ .

- Encuentra  $a \in \mathbb{R}$  para que

$$u(x, y, t) = e^{at} \sin(\pi x) \sin(\pi y) \quad t > 0, (x, y) \in (0, 1)^2$$

sea una solución de (1)

**Solución:**

Para hallar el valor de  $a$  que hace que la función dada sea solución del problema (1) no hay más que introducirla en la  $EDP^1$ , despejar  $a$  y comprobar que cumple las condiciones de contorno. Hagámoslo.

Primero introducimos la solución dada en la  $EDP$ :

$$ae^{at} \sin(\pi x) \sin(\pi y) = -2\pi^2 e^{at} \sin(\pi x) \sin(\pi y) + \pi^2 e^{at} \sin(\pi x) \sin(\pi y)$$

Por lo que igualando términos llegamos a que, tomando  $a = -\pi^2$ ,  $u$  es solución de la  $EDP$ . Ahora comprobemos que verifica las condiciones de contorno:

$$\begin{aligned} u(x, y, 0) &= e^{-\pi^2 \cdot 0} \sin(\pi x) \sin(\pi y) = \sin(\pi x) \sin(\pi y) \\ u(x, 0, t) &= e^{-\pi^2 t} \sin(\pi x) \sin(\pi \cdot 0) = 0 \\ u(x, 1, t) &= e^{-\pi^2 t} \sin(\pi x) \sin(\pi) = 0 \\ u(0, y, t) &= e^{-\pi^2 t} \sin(\pi \cdot 0) \sin(\pi y) = 0 \\ u(1, y, t) &= e^{-\pi^2 t} \sin(\pi) \sin(\pi y) = 0 \end{aligned}$$

Concluyendo así el problema.

- Demuestra que el problema (1) no está bien puesto ya que no tiene dependencia continua del dato inicial. Para ello, dado  $T > 0$  y  $\epsilon > 0$ , encuentra  $f \in C^\infty((0, 1)^2)$  tal que  $|f(x, y)| < \epsilon$  para todo  $(x, y) \in (0, 1)^2$  y la solución del problema:

$$\begin{cases} v_t(x, y, t) = 2v_{xx}(x, y, t) - v_{yy}(x, y, t) & t > 0, (x, y) \in (0, 1)^2 \\ v(x, y, 0) = \sin(\pi x) \sin(\pi y) + f(x, y) & (x, y) \in (0, 1)^2 \\ v(x, 0, t) = v(x, 1, t) = v(0, y, t) = v(1, y, t) = 0 & t > 0, (x, y) \in (0, 1)^2 \end{cases} \quad (2)$$

cumple que  $\sup \{|v(x, y, T) - u(x, y, T)| : (x, y) \in (0, 1)^2\} > 1$ .

**Solución:**

Consideremos la familia de funciones  $f_n \in C^\infty((0, 1)^2)$ ,  $n \in \mathbb{N}$ :

$$f_n(x, y) = \frac{\epsilon}{2} \sin(n\pi x) \sin(2n\pi y) \quad n \in \mathbb{N}$$

En primer lugar comprobamos la hipótesis  $|f_n(x, y)| = \frac{\epsilon}{2} |\sin(n\pi x) \sin(2n\pi y)| < \epsilon \quad \forall n \in \mathbb{N}, (x, y) \in (0, 1)^2$ . Veamos ahora que la familia de funciones:

$$v_n(x, y, t) = e^{-\pi^2 t} \sin(\pi x) \sin(\pi y) + \frac{\epsilon}{2} e^{2n^2 \pi^2 t} \sin(n\pi x) \sin(2n\pi y) \quad n \in \mathbb{N}^2$$

<sup>1</sup> Nótese que la solución propuesta en el enunciado es de clase  $C^\infty([0, 1]^2)$ , por lo que no aparecen problemas de derivabilidad.

<sup>2</sup> Cada solución  $v_n$  se obtiene buscando soluciones de la forma  $v_n(x, y, t) = u(x, y, t) + f_n(x, y)g(t)$ , y pedimos que  $g(t)$  sea una función derivable en  $t > 0$ . En ese punto no es más que resolver un problema de Cauchy, pues las condiciones de contorno se obtienen trivialmente.

son soluciones del problema. Observamos es que el primer término corresponde con la solución del problema (1) obtenida en el apartado anterior, y la segunda parte aparece al introducir las funciones  $f_n$  en las condiciones de contorno. Comprobemos ahora que  $v_n$  es solución del problema (2):<sup>3</sup>

$$n^2 \pi^2 \epsilon e^{2n^2 \pi^2 t} \sin(n\pi x) \sin(2n\pi y) = -n^2 \pi^2 \epsilon e^{2n^2 \pi^2 t} \sin(n\pi x) \sin(2n\pi y) + 2n^2 \pi^2 \epsilon e^{2n^2 \pi^2 t} \sin(n\pi x) \sin(2n\pi y)$$

Nótese que solo lo hemos comprobado para el segundo sumando, pues los operadores derivada parcial son lineales. Ahora comprobamos que  $v$  verifica las condiciones de contorno impuestas:

$$\begin{aligned} v_n(x, y, 0) &= e^{-\pi^2 \cdot 0} \sin(\pi x) \sin(\pi y) + \frac{\epsilon}{2} e^{2n^2 \pi^2 \cdot 0} \sin(n\pi x) \sin(2n\pi y) = \sin(\pi x) \sin(\pi y) + f_n(x, y) \\ v_n(x, 0, t) &= e^{-\pi^2 t} \sin(\pi x) \sin(\pi \cdot 0) + \frac{\epsilon}{2} e^{2n^2 \pi^2 t} \sin(n\pi x) \sin(2n\pi \cdot 0) = 0 \\ v_n(x, 1, t) &= e^{-\pi^2 t} \sin(\pi x) \sin(\pi) + \frac{\epsilon}{2} e^{2n^2 \pi^2 t} \sin(n\pi x) \sin(2n\pi) = 0 \\ v_n(0, y, t) &= e^{-\pi^2 t} \sin(\pi \cdot 0) \sin(\pi y) + \frac{\epsilon}{2} e^{2n^2 \pi^2 t} \sin(n\pi \cdot 0) \sin(2n\pi y) = 0 \\ v_n(1, y, t) &= e^{-\pi^2 t} \sin(\pi) \sin(\pi y) + \frac{\epsilon}{2} e^{2n^2 \pi^2 t} \sin(n\pi) \sin(2n\pi y) = 0 \end{aligned}$$

Por lo que, efectivamente,  $v$  es una solución de (2). Por último, debemos comprobar la desigualdad que se nos indica. Sean  $\epsilon, T > 0$ :

$$\sup \{ |v_n(x, y, T) - u(x, y, T)| : (x, y) \in (0, 1)^2 \} = \sup_{(x, y) \in (0, 1)^2} \left\{ e^{2n^2 \pi^2 T} |f_n(x, y)| \right\} \equiv S_n$$

Ahora bien:

$$\sup_{n \in \mathbb{N}} S_n = \infty > 1$$

Esto es, existen naturales de manera que dicho supremo se hace todo lo grande que queramos, particularmente mayor que 1, tomando un  $n$  suficientemente grande (esencialmente por el dominio de la exponencial sobre un conjunto de supremos que siempre son mayores que 0).

## 2. Ejercicio 2

- Considerar las diferencias finitas de la forma forward:

$$\Delta_h f(x) = \frac{a_0 f(x) + a_1 f(x+h) + a_2 f(x+2h) + a_3 f(x+3h)}{h} \quad (3)$$

**Determinar los valores de  $a_0, a_1, a_2, a_3 \in \mathbb{R}$  que hagan que la derivada  $\Delta_h f(x)$  sea una discretización de  $f'(x)$  del mayor orden posible.**

¡PREGUNTAR LO DEL ORDEN MAYOR QUÉ ES EXACTAMENTE

Como necesitamos una expresión que involucre a  $f(x), f(x+h), f(x+2h), f(x+3h)$ , vamos a hacer un desarrollo en serie de Taylor en las mismas con  $h \approx 0$ , hasta el tercer término en la derivada, pues es hasta donde necesitaremos para obtener un sistema compatible determinado. Veámoslo:

$$\begin{cases} f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4) \\ f(x+2h) = f(x) + 2hf'(x) + 2h^2 f''(x) + \frac{4}{3}h^3 f'''(x) + \mathcal{O}(h^4) \\ f(x+3h) = f(x) + 3hf'(x) + \frac{9}{2}h^2 f''(x) + \frac{9}{2}h^3 f'''(x) + \mathcal{O}(h^4) \end{cases} \quad (4)$$

En este punto no hay más que sustituir en la ecuación (3) y agrupar términos para conseguir:

$$\Delta_h f(x) = \frac{(a_0 + a_1 + a_2 + a_3)f(x) + (a_1 + 2a_2 + 3a_3)hf'(x) + (\frac{1}{2}a_1 + 2a_2 + \frac{9}{2}a_3)h^2 f''(x) + (\frac{1}{6}a_1 + \frac{4}{3}a_2 + \frac{9}{2}a_3)h^3 f'''(x)}{h}$$

Como queremos despejar el valor de  $f'(x)$ , debemos imponer que los coeficientes que multiplican a todos los términos que no sean el de la primera derivada sean nulos, y que que se tenga a que el que multiplique a la misma sea 1 para

<sup>3</sup>1 Nuevamente nos topamos con una función  $C^\infty([0, 1]^2)$ .

obtener  $\Delta_h f(x) = f'(x)^4$ . Si imponemos dichas condiciones obtenemos el sistema:

$$\begin{cases} a_0 + a_1 + a_2 + a_3 = 0 \\ a_1 + 2a_2 + 3a_3 = 1 \\ \frac{1}{2}a_1 + 2a_2 + \frac{9}{2}a_3 = 0 \\ \frac{1}{6}a_1 + \frac{4}{3}a_2 + \frac{9}{2}a_3 = 0 \end{cases}$$

En este punto tan solo debemos resolver el sistema, se puede hacer de cualquier manera, con programas de cálculo simbólico o con el propio python. En cualquier caso los resultados obtenidos son:

$$a_0 = -\frac{11}{6}, \quad a_1 = 3, \quad a_2 = -\frac{3}{2}, \quad a_3 = \frac{1}{3}$$

- **Considerar las diferencias centradas de la forma:**

$$\delta_h f(x) = \frac{b_{-2}f(x-2h) + b_{-1}f(x-h) + b_1f(x+h) + b_2f(x+2h)}{h} \quad (5)$$

**Determinar los valores de  $b_{-2}$ ,  $b_{-1}$ ,  $b_1$ ,  $b_2 \in \mathbb{R}$  que hagan que la derivada  $\delta_h f(x)$  sea una discretización de  $f'(x)$  del mayor orden posible.**

Este apartado se resuelve procediendo de manera totalmente análoga al apartado anterior. Planteamos los desarrollos de Taylor que nos interesan:

$$\begin{cases} f(x-2h) = f(x) - 2hf'(x) + 2h^2f''(x) - \frac{4}{3}h^3f'''(x) + \mathcal{O}(h^4) \\ f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2f''(x) - \frac{1}{6}h^3f'''(x) + \mathcal{O}(h^4) \\ f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(x) + \frac{1}{6}h^3f'''(x) + \mathcal{O}(h^4) \\ f(x+2h) = f(x) + 2hf'(x) + 2h^2f''(x) + \frac{4}{3}h^3f'''(x) + \mathcal{O}(h^4) \end{cases} \quad (6)$$

Sustituyendo en la ecuación (5) se obtiene:

$$\delta_h f(x) = \frac{(b_{-2}+b_{-1}+b_1+b_2)f(x) + (-2b_{-2}-b_{-1}+b_1+2b_2)hf'(x) + (2b_{-2}+\frac{1}{2}b_{-1}+\frac{1}{2}b_1+2b_2)h^2f''(x) + (-\frac{4}{3}b_{-2}-\frac{1}{6}b_{-1}+\frac{1}{6}b_1+\frac{4}{3}b_2)h^3f'''(x)}{h}$$

Planteamos el sistema con las mismas restricciones que el apartado anterior:

$$\begin{cases} b_{-2} + b_{-1} + b_1 + b_2 = 0 \\ -2b_{-2} - b_{-1} + b_1 + 2b_2 = 1 \\ 2b_{-2} + \frac{1}{2}b_{-1} + \frac{1}{2}b_1 + 2b_2 = 0 \\ -\frac{4}{3}b_{-2} - \frac{1}{6}b_{-1} + \frac{1}{6}b_1 + \frac{4}{3}b_2 = 0 \end{cases}$$

Donde obtenemos como soluciones:

$$b_{-2} = \frac{1}{12}, \quad b_{-1} = -\frac{2}{3}, \quad b_1 = \frac{2}{3}, \quad b_2 = -\frac{1}{12}$$

- **Considerad  $f = \cos(x)$ . Construid un programa y calculad numéricamente  $f'(x)$  en el intervalo  $x \in [0, 2\pi]$  utilizando las discretizaciones  $\Delta_h, \delta_h$  que acabáis de obtener. Calculad el error de vuestra derivada en ambos casos para diferentes valores de  $h$  y discutid si el error se ajusta a las predicciones teóricas. Recomendación: pintar el error en escala logarítmica.**

**Solución:**

Al ser este apartado de la práctica especialmente largo, vamos a aproximarnos a él por partes.

<sup>42</sup> En el límite conforme  $h$  tiende a 0, por supuesto. Nótese que hemos despreciado todos los términos de las series de orden mayor que 3.

## Cálculo de las derivadas forward y centrada

Vamos a exponer primero el código utilizado, para posteriormente comentarlo:

```

1  ##EJERCICIO 2 E)
2  '''PARTE 1- CALCULO DE DERIVADASIMPORTAMOS LAS LIBRERIAS QUE NOS VAN A SER UTILES'''
3  import numpy
4  from numpy import *
5  import matplotlib.pyplot as plt
6  '''DEFINIMOS LA FUNCION EVALUAR EN LA FUNCION CON LA QUE VAMOS A TRABAJAR'''
7  def evaluar_coseno(t):
8      f=cos(t)
9      return f
10  '''Y EN SU DERIVADA'''
11  def evaluar__menos_seno(t):
12      f= -sin(t)
13      return f
14  '''AHORA SI, EXPONEMOS EL CODIGO DE LA DERIVADA FORWARD'''
15  def derivada_forward(t,f):
16      '''INICIALIZAMOS UNA LISTA QUE SERA NUESTRA DERIVADA'''
17      derivada=[]
18      '''h ES EL SALTO DE UN VECTOR QUE SUPONEMOS EQUIESPACIADO '''
19      h=t[1]-t[0]
20      '''EXTENDEMOS NUESTRO VECTOR DE DERIVADAS, MIRAR .TEX PARA RAZONAMIENTO'''
21      t=concatenate((t,array([t[0],t[1],t[2]])))
22      f=concatenate((f,array([f[0],f[1],f[2]])))
23      '''OBTENEMOS EL VECTOR DERIVADA MOVIENDONOS CON UN FOR'''
24      for i in range (0,len(t)-3):
25          derivada.append((-11/6*f[i]+3*f[i+1]-1.5*f[i+2]+1/3*f[i+3])/h)
26      '''EXTRAEMOS EL VECTOR DERIVADA'''
27      return derivada
28  def main():
29      '''CREAMOS NUESTRO VECTOR TEMPORAL Y SU CORRESPONDIENTE IMAGEN'''
30      t=linspace(0,2*pi,100)
31      f=evaluar_coseno(t)
32      '''CREAMOS UN VECTOR DE LA MISMA LONGITUD CON LA DERIVADA VERDADERA'''
33      derivada_v=evaluar__menos_seno(t)
34      '''EJECUTAMOS LA FUNCION DERIVADA FORWARD'''
35      derivada=derivada_forward(t,f)
36      '''PLOTEAMOS AMBAS FUNCIONES PARA COMPARARLAS'''
37      plt.scatter(t,derivada,label='Derivada forward coseno',color='orange')
38      plt.plot(t,derivada_v,label='Derivada verdadera coseno')
39      plt.legend()
40      plt.xlabel('t')
41      plt.ylabel('$-\sin(t)$')
42      plt.show()
43  if __name__ == '__main__':
44      main()

```

Lo único reseñable de este código que se adjunta es cómo se han conseguido las condiciones de contorno en los extremos y cómo se ha realizado la computación de los mismos. Respecto a lo primero, no hay más que darse cuenta de que, al tratarse de funciones periódicas:

$$f'(t+h) = f'(t+h-T), \quad t, h \in \mathbb{R}, T > 0 \quad (7)$$

Esto nos permite resolver un problema que teníamos, pues si nos damos cuenta, la derivada forward a priori no permite calcular la derivada de ningún término posterior al cuarto empezando por la cola. Sin embargo, si utilizamos la ecuación (7), nos damos cuenta de que los tres términos que nos faltan los podemos extraer de los tres primeros términos del vector, por lo que basta con concatenarlos al final para poder sacar la derivada en dichos términos. Esto

es precisamente lo que hacemos con los vectores  $f$  y  $t$  del código. Después, simplemente no calculamos la derivada en los mismos extrayéndolos del bucle *for*. Presentamos el *plot* obtenido al representar la derivada forward frente a la derivada real:

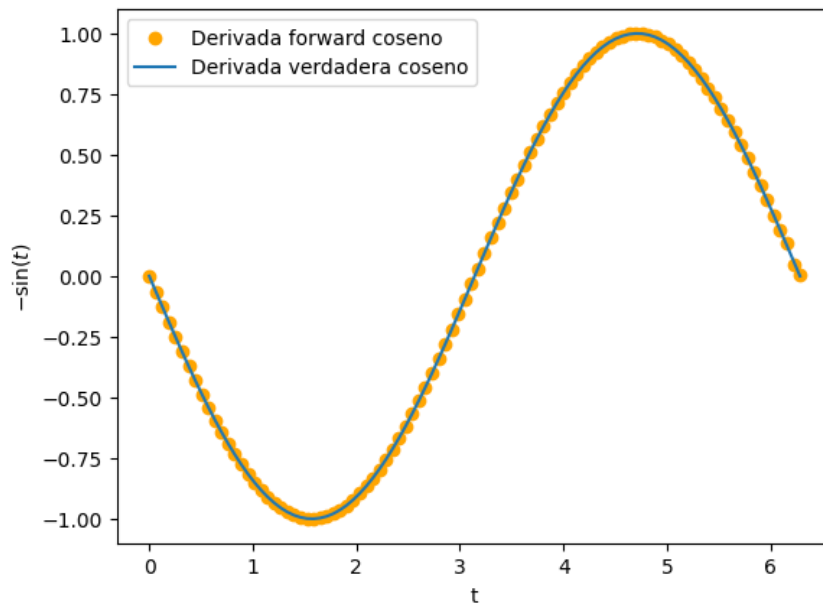


Imagen 1: Plot de la derivada forward frente a la derivada real

El desarrollo de la derivada centrada es totalmente análogo al de la derivada forward, con una salvedad que se comentará después de exponer el código:

```

1 def derivada_centrada(t,f):
2     derivada=[]
3     h=t[1]-t[0]
4     '''EXTENDEMOS NUESTROS VECTORES POR DELANTE Y DETRAS PARA PODER DERIVAR'''
5     t=concatenate((t[len(t)-2:],t,array([t[0],t[1]])))
6     f=concatenate((f[len(f)-2:],f,array([f[0],f[1]])))
7     for i in range(2,len(t)-2):
8         derivada.append((1/12*f[i-2]-2/3*f[i-1]+2/3*f[i+1]-1/12*f[i+2])/h)
9     return derivada
10
11 def main():
12     t=linspace(0,2*pi,100)
13     f=evaluar_coseno(t)
14     derivada_v=evaluar_menos_seno(t)
15     derivada=derivada_centrada(t,f)
16     plt.scatter(t,derivada,label='Derivada centrada coseno',color='orange')
17     plt.plot(t,derivada_v,label='Derivada verdadera coseno')
18     plt.legend()
19     plt.xlabel('t')
20     plt.ylabel('$-\sin(t)$')
21     plt.show()
22
23 if __name__ == '__main__':
24     main()

```

Si nos fijamos en la expresión de la derivada centrada, observamos que se nos presenta el mismo problema que en la forward, pero con una salvedad: en este caso los problemas de cálculo de la derivada se nos presentan tanto al principio como al final del vector de tiempos. Es por eso que, recurriendo nuevamente a la expresión (7), no tenemos que hacer más que repetir el proceso de concatenación de vectores realizado en el apartado anterior, pero en este caso añadiendo los dos últimos elementos al principio, y los primeros dos al final; excuyéndolos luego del cálculo de la derivada, igual que en la derivada forward. Para terminar esta subsección del apartado, presentamos de igual manera un plot que

enfrenta a la derivada mediante diferencias finitas con la derivada real:

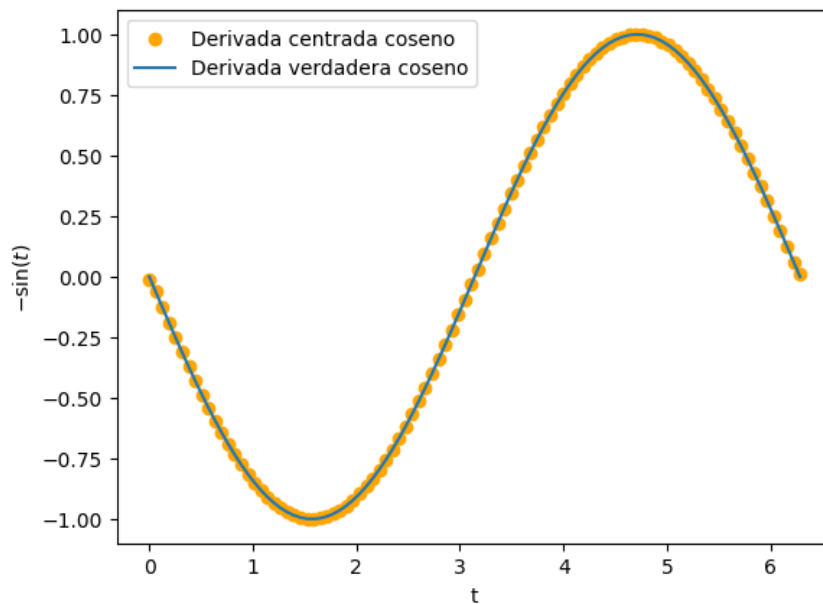


Imagen 2: Plot de la derivada centrada frente a la derivada real

## Cálculo de errores

Para terminar la sección presentamos una muestra de los errores producido al utilizar ambos tipos de derivadas. Este proceso permite comprobar o desmentir la eficiencia del cálculo de derivadas mediante estos dos métodos con dos formas distintas.

Presentamos ahora el código utilizado y lo desgranamos a continuación:

```

1  '''PARTE 2- CALCULO DE ERRORES'''
2  def calculo_errores(n_h):
3      '''INICIALIZAMOS UNA LISTA DE ERRORES PARA CADA DERIVADA, Y UNA LISTA DE
4          SALTOS h'''
5      error_f=[]
6      error_c=[]
7      h=[]
8      '''UTILIZAMOS UN VECTOR FOR PARA MOVERNOS EN EL VECTOR QUE MODIFICA h'''
9      for i in range (0,len(n_h)):
10         '''EN CADA ITERACION AUMENTA EL NUMERO DE TIEMPOS, Y DISMINUYE h'''
11         t=linspace(0,2*pi,10*int(n_h[i]))
12         '''METEMOS h EN LA LISTA EN CADA ITERACION'''
13         h.append(t[1]-t[0])
14         '''EJECUTAMOS LOS PROGRAMAS DE DERIVADAS'''
15         f=evaluar_coseno(t)
16         derivada_f=derivada_forward(t,f)
17         derivada_c=derivada_centrada(t,f)
18         derivada_v=evaluar_menos_seno(t)
19         '''CALCULAMOS LOS ERRORES COMO LA MEDIA DEL ERROR ABSOLUTO, EN CADA h'''
20         error_f.append(mean(abs(derivada_f-derivada_v)))
21         error_c.append(mean(abs(derivada_c-derivada_v)))
22         '''PINTAMOS LOS ERRORES FRENTE A h'''
23         plt.scatter(h,error_f,label='Error forward',color='orange')
24         plt.scatter(h,error_c,label='Error centrado')
25         plt.legend()
26         plt.xlabel('h')

```



```

26     plt.ylabel('Error')
27     plt.show()
28     '''PINTAMOS LOS ERRORES FRENTE A h, AMBOS EN ESCALA LOGARITMICA'''
29     plt.scatter(log(h), log(error_f), label='Error logar tmico forward', color='
orange')
30     plt.scatter(log(h), log(error_c), label='Error logar tmico centrado')
31     plt.legend()
32     plt.xlabel('$\log(h)$')
33     plt.ylabel('$\log(Error)$')
34     plt.show()
35
36 def main():
37     '''CREAMOS EL VECTOR QUE NOS VA A DETERMINAR EL NUMERO DE ELEMENTOS DEL VECTOR
        TIEMPO,
38     Y h COMO CONSECUCENCIA'''
39     n_h=linspace(5000,10000,30)
40     calculo_erros(n_h)
41 if __name__ == '__main__':
42     main()

```

Para entender el código expuesto vamos primero a ilustrar un concepto de discretización de la derivada. Según la definición de derivada de una función en un punto:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (8)$$

Esto implica particularmente, que:

$$f'(x) = \lim_{n \rightarrow \infty} \frac{f(x + \frac{1}{n}) - f(x)}{1/n} \quad (9)$$

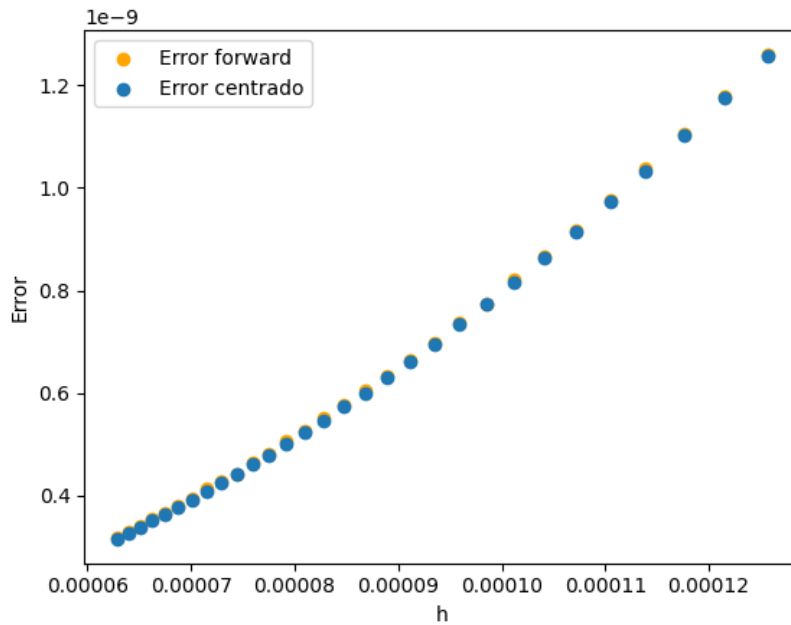
La elección de las ecuaciones (8) y (9) no es arbitraria, de hecho utilizamos un concepto subyacente a las mismas. Si nos fijamos en la función de cálculo de errores, su argumento de entrada es un *array*  $n_h$  que se caracteriza por contener un número creciente de números, que serían como nuestro  $n$  en la expresión (9). Que la  $n$  aumente, implica que la  $h$  disminuya, y asintóticamente se obtiene lo que nos indican estas ecuaciones.<sup>5</sup>

Con todo esto queremos llegar a que, al hacer  $n$  infinitamente grande,  $h$  se hará infinitesimalmente pequeño, y, en el límite, producirá que las derivadas centrada y forward coincidan con la derivada real. Por supuesto, esto es análisis numérico, por lo que no podemos aspirar a contar con un conjunto de cardinal infinito numerable, y por lo tanto nos vamos a limitar a observar el crecimiento en el error con  $h$ .<sup>6</sup>

Los resultados obtenidos son:

<sup>5</sup>2 Aunque las expresiones que hemos aportado para ejemplificar el decrecimiento en el error de la derivada no son las correctas (pues hemos partido de la definición de derivada, y no de la forward y la centrada), el principio se ilustra de igual manera.

<sup>6</sup>2 Es por esto que no presentamos el valor numérico de los errores, pues no aportan más significado que el gráfico.



<sup>7</sup> Imagen 3: Representación del error obtenido frente a  $h$

Ahora bien, en el enunciado se nos indica que probemos a tomar escala logarítmica. La ventaja que tiene tomar logaritmos en los dos argumentos de un plot, es que, en caso de que la relación entre ambas variables sea polinómica y de la forma  $y = x^p$ , el logaritmo<sup>8</sup> hace que dicha relación se vuelva razonablemente lineal, y el exponente se convierta en la pendiente de la recta (la ordenada en el origen no tiene ningún interés en este caso). La representación queda:

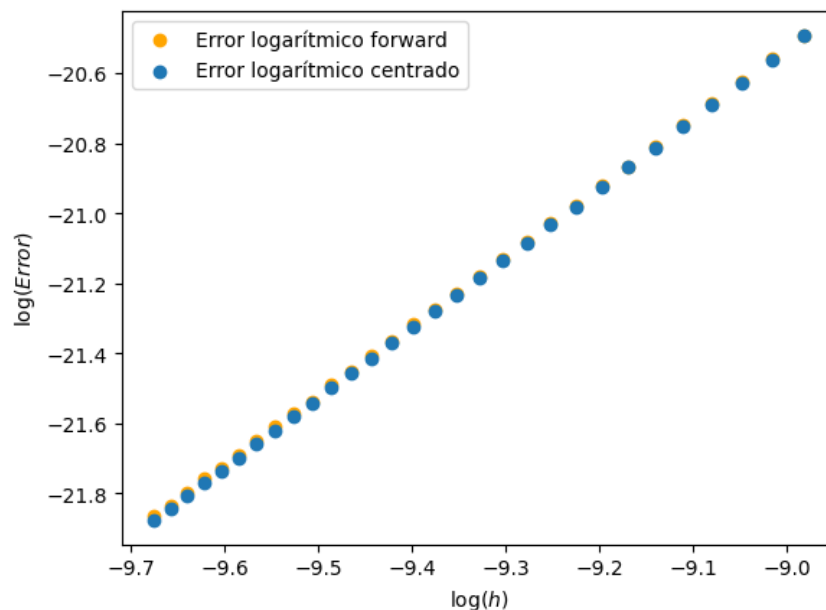


Imagen 4: Representación logarítmica del error frente al logaritmo de  $h$

Además, por hipótesis se debería cumplir que:

$$(h, error(h)) \xrightarrow{h \rightarrow 0} -(\infty, \infty)$$

Y podemos afirmar que se cumple.

PREGUNTAR AL TIO SI HAY QUE HACER UN FIT O NO PARA HALLAR LA PENDIENTE.

<sup>7</sup>2 Los errores de las dos derivadas se solapan pues son muy similares.

<sup>8</sup>2 A priori no importa la base elegida

### 3. Ejercicio 3

**Ejercicio 3.** Considere la siguiente tabla y responde a las cuestiones posteriores:

$\alpha$ (grados)	0	30	60	90	120	150	180	210	240	270	300	330
$\delta$ (minutos)	408	89	-66	10	338	807	1238	1511	1583	1462	1183	804

Tabla 1: Datos trayectoria asteroide Pallas mediante las tablas del Barón von Zach

- **Asume la existencia de una función periódica  $\delta = f(\alpha)$ . Aproxima  $f(\alpha)$  mediante los datos de la tabla y el algoritmo rFFT. ¿Cuántos coeficientes de la serie de Fourier correspondientes podemos obtener? ¿A cuántas frecuencias corresponden?**

**Solución:**

En primer lugar comentamos que la primera suposición que debemos hacer es que la función  $f(\alpha)$  que se nos presenta es  $2\pi$  radianes ( $360^\circ$ ) periódica (al representarla parece claro). Sin embargo, en el código que se presenta a continuación, se utilizan funciones de un periodo arbitrario  $T$ . Adjuntamos el código utilizado y luego lo comentamos:

```

1  # APARTADO F
2
3  '''
4  DEFINIMOS LA FUNCION QUE NOS VA A PERMITIR CALCULAR LOS COEFICIENTES DE FOURIER.
5  SU UNICO ARGUMENTO DE ENTRADA ES UN VECTOR DE IMAGENES UNIFORMEMENTE DISTRIBUIDAS
6  EN EL PERIODO.
7  '''
8  def coeficientes_fourier(delta):
9      '''LA FUNCION rFFT NOS DA LOS COEFICIENTES COMPLEJOS DE FOURIER'''
10     c_i = fft.rfft(delta, norm='forward')
11     '''LOS COEFICIENTES COSENOIDALES {a_i}, Y SINUSOIDALES {b_i} LOS OBTENEMOS
12     MEDIANTE SIMPLES
13     RELACIONES ALGEBRAICAS, USANDO QUE  $c_i = \frac{a_i - ib_i}{2}$ '''
14     a_i = c_i + conj(c_i)
15     b_i = (c_i - conj(c_i)) * 1j
16     '''ELIMINAMOS b_i[0], PUES SABEMOS QUE c_i[0] = a_i[0/2], e.d.  $c_i[0] \in \mathbb{R}$ '''
17     b_i = b_i[1:]
18     '''HACEMOS QUE LA FUNCION NOS DEVUELVA LOS COEFICIENTES'''
19     return a_i, b_i, c_i
20
21 def main():
22     '''PRIMERO DEFINIMOS LOS VECTORES ALPHA Y DELTA QUE VAMOS A UTILIZAR'''
23     alpha = linspace(0, 330, 12)
24     delta = array([408, 89, -66, 10, 338, 807, 1238, 1511, 1583, 1462, 1183, 804])
25     '''AHORA REPRESENTAMOS LOS PUNTOS OBTENIDOS CON LA FUNCION SCATTER PARA
26     POSTERIORMENTE
27     COMPARARLO CON LOS RESULTADOS QUE OBTENGAMOS'''
28     plt.scatter(alpha, delta, label='Datos Bar n von Zach')
29     plt.xlabel('Ascension recta (Grados)')
30     plt.ylabel('Declinacion (Minutos)')
31     plt.legend()
32     plt.show()
33     a_i, b_i, c_i = coeficientes_fourier(delta)
34     '''AHORA EXPONEMOS LOS COEFICIENTES DE FOURIER OBTENIDOS'''
35     print('Los coeficientes complejos de la serie de Fourier son ' + str(c_i))
36     print('Los coeficientes cosenoidales son ' + str(a_i))
37     print('Los coeficientes sinusoidales son ' + str(b_i))
38
39 if __name__ == '__main__':
40     main()

```

Si ejecutamos el programa obtenemos:

```

1 Los coeficientes complejos de la serie de Fourier son [ 7.80583333e+02+0.00000000e+00j
2 -2.05507183e+02+3.60113946e+02j
3 2.17083333e+01+1.08253175e+00j -2.16666667e+00-2.75000000e+00j
4 -5.41666667e-01+5.05181486e-01j 1.73850033e-01+1.36053580e-01j
5 8.33333333e-02+0.00000000e+00j]
6 Los coeficientes cosenoidales son [ 1.56116667e+03+0.j -4.11014367e+02+0.j 4.34166667
7 e+01+0.j
8 -4.33333333e+00+0.j -1.08333333e+00+0.j 3.47700065e-01+0.j
9 1.66666667e-01+0.j]
10 Los coeficientes sinusoidales son [-7.20227893e+02+0.j -2.16506351e+00+0.j 5.50000000
11 e+00+0.j
12 -1.01036297e+00+0.j -2.72107160e-01+0.j 0.00000000e+00+0.j]

```

Y la imagen:

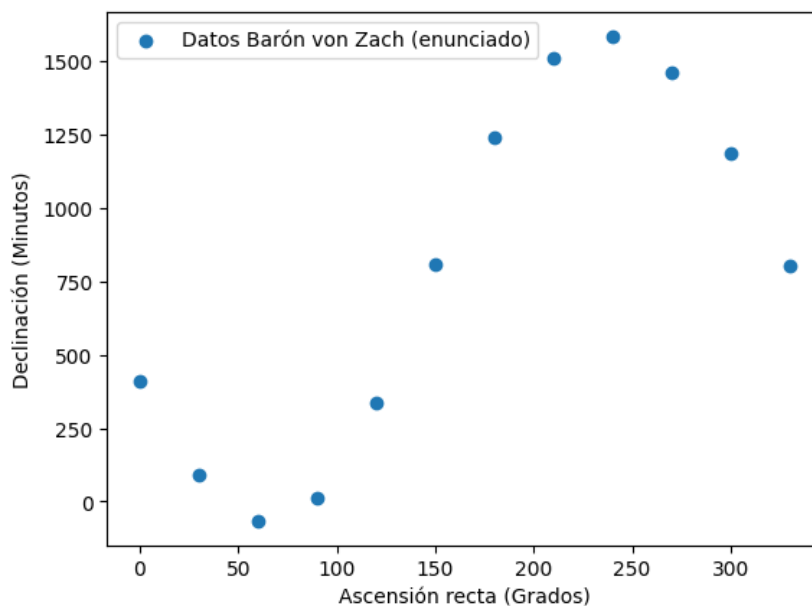


Imagen 5: Plot de los datos del enunciado.

Vamos a comentar los resultados obtenidos. Si nos fijamos en los resultados obtenidos en la consola, hemos obtenido un total de 7 elementos del vector  $c_i$ . Sin embargo, utilizando la simetría en los coeficientes de Fourier :

$$c_k = c_{-k}^*, \quad k \geq 1$$

Y utilizando que, como se menciona en el código, el primer coeficiente de Fourier (el coeficiente 0) es real, por lo que obtenemos un total de  $6 \times 2 + 1 = 13$  coeficientes de Fourier. Ahora bien, si consideramos que las frecuencias corresponden con aquellos términos no nulos en senos y cosenos, particularmente con los términos que corresponden a sus argumentos:

$$\omega_i = \frac{2\pi k_i}{T}, \quad k_i \in \{i\}_{i=1}^6, \quad T = 360^\circ$$

- **Redondea los coeficientes hasta la segunda cifra decimal y proporciona la aproximación de  $\delta$  en serie de Fourier trigonométrica.**

**Solución:**

Primero introducimos el código para redondear los coeficientes hasta el segundo término:

```

1 # APARTADO G
2 ''' AHORA CONSIDERAMOS LOS COEFICIENTES DE FOURIER OBTENIDOS EN EL APARTADO ANTERIOR Y
   LOS REDONDEAMOS

```

```

3  HASTA EL SEGUNDO TERMINO'''
4  b_i=around(b_i,2)
5  a_i=around(a_i,2)
6  c_i=around(c_i,2)
7
8  if __name__ == '__main__':
9      main()
10 '''AHORA SIMPLEMENTE UTILIZAMOS LA DEFINICION DE LA SERIE DE FOURIER Y LA EXPRESAMOS:
    '''
11 print('Los coeficientes cosenoidales son :'+str(a_i))
12 print('Los coeficientes sinusoidales son :'+str(b_i))

```

Y obtenemos por pantalla:

```

1  Los coeficientes cosenoidales son :[ 1.56117e+03+0.j -4.11010e+02+0.j  4.34200e+01+0.j
   -4.33000e+00+0.j
2  -1.08000e+00+0.j  3.50000e-01+0.j  1.70000e-01+0.j]
3  Los coeficientes sinusoidales son :[-7.2023e+02+0.j -2.1700e+00+0.j  5.5000e+00+0.j
   -1.0100e+00+0.j
4  -2.7000e-01+0.j  0.0000e+00+0.j]

```

Y la serie de Fourier queda:

$$f(\alpha) = \frac{1561,17}{2} - 411,01 \cos\left(\frac{\pi}{180}\alpha\right) - 720,23 \sin\left(\frac{\pi}{180}\alpha\right) + 43,42 \cos\left(\frac{\pi}{90}\alpha\right) - 2,17 \sin\left(\frac{\pi}{90}\alpha\right) - 4,33 \cos\left(\frac{\pi}{60}\alpha\right) + 5,50 \sin\left(\frac{\pi}{60}\alpha\right) - 1,08 \cos\left(\frac{\pi}{45}\alpha\right) - 1,01 \sin\left(\frac{\pi}{45}\alpha\right) + 0,35 \cos\left(\frac{\pi}{36}\alpha\right) - 0,27 \sin\left(\frac{\pi}{36}\alpha\right) + 0,17 \cos\left(\frac{\pi}{30}\alpha\right) + 0 \sin\left(\frac{\pi}{30}\alpha\right)$$

- Representa la función  $f(\alpha)$  obtenida.

**Solución:**

Una vez ya obtenidos coeficientes de Fourier podemos sustituir en la expresión de la serie. En nuestro caso lo vamos a hacer en representación exponencial. Como siempre adjuntamos el código y luego comentamos:

```

1  #APARTADO H
2  '''UTILIZAMOS UNA FUNCION QUE TIENE COMO ARGUMENTOS DE ENTRADA
3  UN VECTOR DE TIEMPOS (EN ESTE CASO GRADOS), LOS COEFICIENTES
4  DE LA SERIE DE FOURIER OBTENIDOS ANTERIORMENTE, Y EL PERIODO
5  DE LA FUNCION DE LA QUE TENEMOS QUE INTERPOLAR.'''
6  def evaluar_delta(alpha,c_i,T):
7      evalu=c_i[0]*ones(len(alpha))
8      for i in range(1,len(c_i)):
9          evalu=evalu+c_i[i]*exp(2*pi*1j*i*alpha/T)+conj(c_i[i])*exp(2*pi*1j*(-i
10         )*alpha/T)
11      return evalu
12
13 def main():
14     alpha_largo=linspace(0,330,1000)
15     T=360
16     evalu=evaluar_delta(alpha_largo,c_i,T)
17     '''PRINTEAMOS LOS DATOS DEL ENUNCIADO JUNTO CON LA INTERPOLACION
18     CON MUCHOS DATOS PARA GARANTIZAR LA SUAVIDAD DE LA CURVA'''
19     plt.plot(alpha_largo,evalu,label='Interpolación $\delta$')
20     plt.scatter(alpha,delta,label='Datos enunciado')
21     plt.xlabel('Alpha(grados)')
22     plt.ylabel('Delta (minutos)')
23     plt.legend()
24     plt.show()
25
26 if __name__ == '__main__':

```

25 main()

Presentamos los resultados obtenidos:

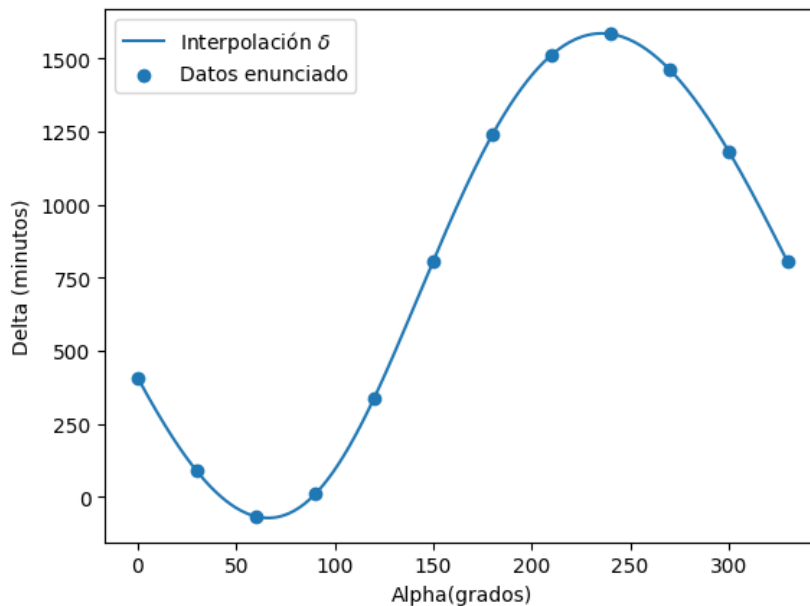


Imagen 6: Plot de los datos del enunciado frente a la interpolación.

En vista a los datos obtenidos (podríamos también calcular el error obtenido interpolando en los datos del enunciado y restándoselos a los de la interpolación, pero no tiene mucho interés), podemos afirmar que los coeficientes se han obtenido con éxito, y que la curva obtenida es, en efecto  $2\pi$  periódica, como habíamos predicho.

- **Calcula la declinación que se espera a  $45^\circ$  de AR.**

**Solución:**

Lo que vamos a hacer es calcular el valor numéricamente y sobrescribirlo sobre la gráfica generada en el apartado anterior. Adjuntamos el código:

```
1 #APARTADO I
2 '''PARA CALCULAR EL VALOR DE LA DECLINACION A 45   SIMPLEMENTE
3 TENEMOS QUE UTILIZAR LA FUNCION CREADA EN EL APARTADO
4 ANTERIOR Y SUSTITUIR POR EL VALOR INDICADO'''
5 def main():
6     alpha_largo=linspace(0,330,1000)
7     T=360
8     evalu=evaluar_delta(alpha_largo,c_i,T)
9     f_45=evaluar_delta(array([45]),c_i,T)
10    '''PRINTEAMOS LOS DATOS DEL ENUNCIADO JUNTO CON LA INTERPOLACION
11    CON MUCHOS DATOS PARA GARANTIZAR LA SUAVIDAD DE LA CURVA'''
12    plt.plot(alpha_largo,evalu,label='Interpolacion $\delta$')
13    plt.scatter(45,f_45,label='Interpolacion a 45 ')
14    print('La interpolacion de la funcion a 45 grados es '+str(f_45))
15    plt.xlabel('Alpha(grados)')
16    plt.ylabel('Delta (minutos)')
17    plt.legend()
18    plt.show()
19 if __name__ == '__main__':
20     main()
```

Y se obtiene  $f(45) = -13,49197083'$ . La gráfica que se obtiene es:

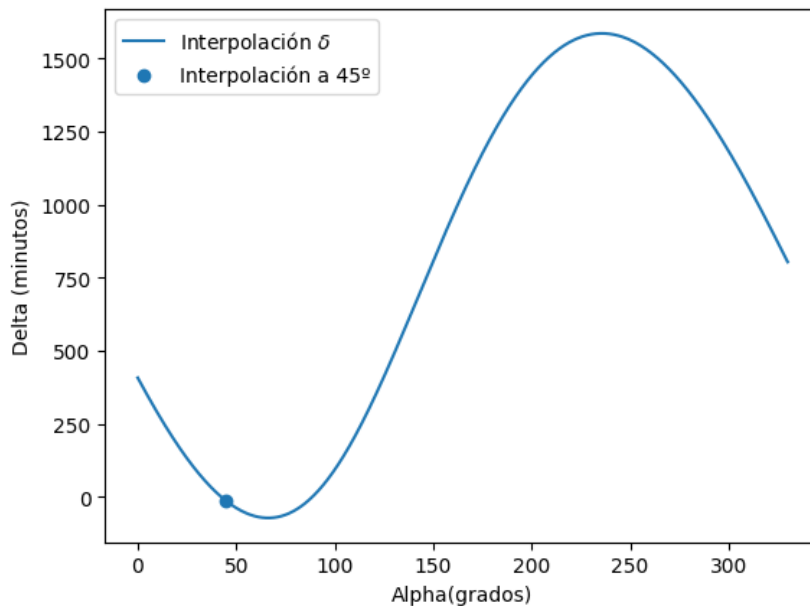


Imagen 7: Plot de los datos de la interpolación anterior frente al valor a 45° de ascensión recta.

- Finalmente, utiliza la serie de Fourier estimada para estimar la tasa de cambio de  $\delta$  en función de  $\alpha$  y representa la función resultante. Así mismo estima el valor de  $f'(45)$ . Se valorará positivamente la resolución de este ejercicio mediante el uso de un algoritmo lo más general posible.

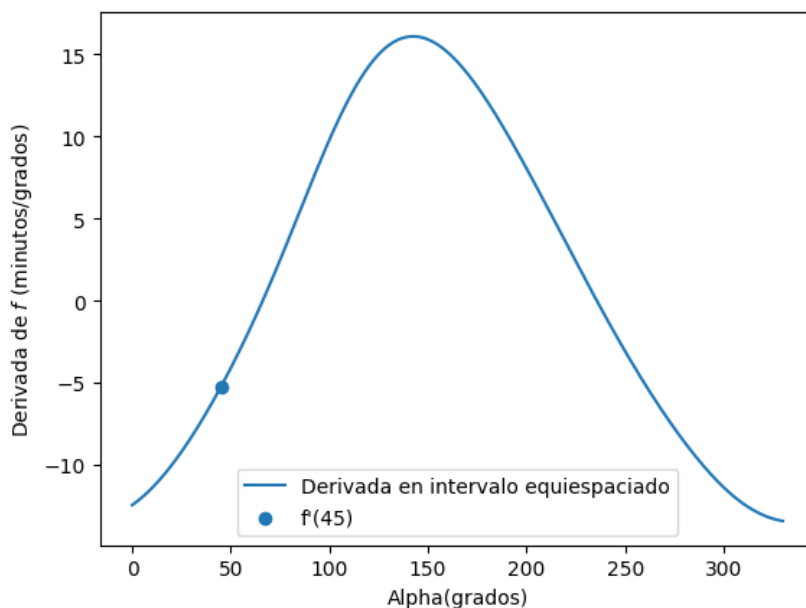
Solución:

```

1 #APARTADO J
2 '''CREAMOS LA FUNCION DERIVADA CON 3 ARGUMENTOS DE ENTRADA, EL VECTOR DE GRADOS, LOS
   COEFICIENTES DE FOURIER Y EL PERIODO DE LA FUNCION.'''
3 def evaluar_delta_prima(alpha,c_i,T):
4     'CREAMOS UN VECTOR DE CEROS PARA INICIALIZAR EL VECTOR EVALUACION DE LA
       DERIVADA'
5     evalu=zeros(len(alpha))
6     'EMPEZAMOS EL BUCLE FOR EN EL PRIMER TERMINO PORQUE LA DERIVADA DE LAS
       CONSTANTES ES NULA'
7     for i in range(1,len(c_i)):
8         evalu=evalu+c_i[i]*(2*pi*1j*i/T)*exp(2*pi*1j*i*alpha/T)+conj(c_i[i])
9         *(2*pi*1j*(-i)/T)*exp(2*pi*1j*(-i)*alpha/T)
10    return evalu
11 def main():
12     '''CREAMOS UN VECTOR DE TIEMPOS QUE NOS PERMITA OBTENER UNA VERSION
       RAZONABLEMENTE SUAVE
13     DE LA FUNCION'''
14     alpha=linspace(0,330,1000)
15     f_prima=evaluar_delta_prima(alpha,c_i,360)
16     plt.plot(alpha,f_prima,label='Derivada en intervalo equiespaciado')
17     f_prima_45=evaluar_delta_prima(array([45]),c_i,360)
18     'PRINTEAMOS EL VALOR DE LA FUNCION EN EL PUNTO DE INTERES'
19     print('El valor de la derivada en 45 es '+str(f_prima_45))
20     plt.scatter([45],f_prima_45,label="f' (45) ")
21     plt.xlabel('Alpha(grados)')
22     plt.ylabel('Derivada de $f$ (minutos/grados)')
23     plt.legend()
24     plt.show()
25 if __name__ == '__main__':
    main()

```

Y obtenemos un valor de la derivada en el punto indicado  $f'(45) = -5,24831183' / \circ$ . Presentamos también un gráfico en el que enseñamos el plot del código:



. Imagen 8: Plot de los datos de la interpolación anterior frente al valor a  $45^\circ$  de ascensión recta.

Se observa que los resultados son coherentes, pues los ceros en la derivada se alcanzan en los extremos relativos de la función original, y el máximo se alcanza en torno al cero de la misma.

Para terminar, simplemente debemos comentar que el ejercicio es todo lo general que puede ser, puesto que la descomposición en serie de Fourier es única; esto es, si se trata de una función periódica esta viene representada unívocamente por sus coeficientes de Fourier. Si bien es cierto que si se introducen todos coeficientes de la serie (en la hipótesis de que esta tenga finitos términos no nulos), la solución obtenida es exacta, en caso de querer un análisis puramente numérico, bastaría con aportar un conjunto de datos de la función sobre el periodo y aplicar la función que nos otorga los coeficientes.

## 4. Anexo

### 4.1. Código

El código de Python que se ha utilizado durante el desarrollo de la práctica es un notebook se puede encontrar en el [siguiente directorio de Github.io](#).