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

Instrucción

A continuación se adjunta un pdf explicatorio de la práctica. El código de Python que se ha utilizado durante el desarrollo de la práctica es un notebook se puede encontrar en el siguiente repositorio de Github.io.

1. Ejercicio 1

Ejercicio 1. Considere el siguiente problema de evolución:

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

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**

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

sea una solución de (3)

Solución:

Para hallar el valor de a que hace que la función dada sea solución del problema (3) 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:

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

Concluyendo así el problema.

- **Demuestra que el problema** (3) **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:**

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

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

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

Solución:

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

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

En primer lugar comprobamos la hipótesis $|f_n(x,y)| = \frac{\epsilon}{2}|\sin(n\pi x)\sin(2n\pi y)| < \epsilon \quad \forall n \in \mathbb{N}, (x,y) \in (0,1)^2$. Dadas dichas $f_n(x,y)$ vamos a buscar soluciones de la forma $v_n(x,y,t) = u(x,y,t) + g_n(t)f_n(x,y)$. Si nos damos cuenta, esto convierte nuestro problema de $EDPs$ con condiciones de contorno en un problema de valor inicial (pues las condiciones de contorno se verifican de manera trivial, se adjuntan después). El problema en cuestión resulta:

$$\begin{cases} g_n'(t) = -2n^2\pi^2 g_n(t) + 4n^2\pi^2 g_n(t) = 2n^2\pi^2 g_n(t) & t > 0 \\ g_n(0) = 1 \end{cases} \tag{3}$$

Veamos ahora que la familia de funciones:

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

son soluciones del problema. Observamos es que el primer término corresponde con la solución del problema (3) 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):[2]

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

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

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

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

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

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:**

$$\Delta_h f(x) = \frac{a_0 f(x) + a_1 f(x+h) + a_2 f(x+2h) + a_3 f(x+3h)}{h} \tag{4}$$

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

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

Solución:

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:

$$\begin{cases} f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(x) + \frac{1}{6}h^3f'''(x) + \mathcal{O}(h^4) \\ f(x+2h) = f(x) + 2hf'(x) + 2h^2f''(x) + \frac{4}{3}h^3f'''(x) + \mathcal{O}(h^4) \\ f(x+3h) = f(x) + 3hf'(x) + \frac{9}{2}h^2f''(x) + \frac{9}{2}h^3f'''(x) + \mathcal{O}(h^4) \end{cases} \tag{5}$$

En este punto no hay más que sustituir en la ecuación (4) y agrupar términos para conseguir:

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

Como queremos despejar el valor de $f'(x)$, debemos imponer que los coeficientes que multiplcan 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 obtener $\Delta_h f(x) = f'(x)$[3]. 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:

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

Ahora bien, ¿como podemos saber si el orden tomado es el máximo? Estos números representan el mayor orden que podemos imponer en la derivada sin despreciarla para poder conseguir los coeficientes que se nos indican en el enunciado (para que sea una discretización del mayor orden). Algebraicamente, esto implica encontrar el mayor orden en la derivada tal que el sistema que se obtenga con los coeficientes sea compatible y determinado.
Si nos fijamos en la derivada forward, si intentamos conseguir un término mayor la tercera derivada, nos topamos con que el sistema que tenemos que resolver se vuelve incompatible, pues se añade la ecuación:

$$\frac{1}{24}a_1 + \frac{2}{3}a_2 + \frac{27}{8}a_3 = 0$$

Que no verifican los coeficientes obtenidos anteriormente. Por lo que podemos afirmar que el orden obtenido es 3.

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

$$\delta_h f(x) = \frac{b_{-2}f(x-2h) + b_{-1}f(x-h) + b_1f(x+h) + b_2f(x+2h)}{h} \tag{6}$$

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.

Solución:

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

$$\begin{cases} f(x-2h) = f(x) - 2hf'(x) + 2h^2f''(x) - \frac{4}{3}h^3f'''(x) + \mathcal{O}(h^4) \\ f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2f''(x) - \frac{1}{6}h^3f'''(x) + \mathcal{O}(h^4) \\ f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(x) + \frac{1}{6}h^3f'''(x) + \mathcal{O}(h^4) \\ f(x+2h) = f(x) + 2hf'(x) + 2h^2f''(x) + \frac{4}{3}h^3f'''(x) + \mathcal{O}(h^4) \end{cases} \tag{7}$$

[3] 2 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.

Sustituyendo en la ecuación (6) 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:

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

Nuevamente volvemos a hacernos la pregunta: ¿hemos llegado al mayor orden? Si intentamos meter uno rden a la derivada mayor que tres, nos aparece una ecuación más en coeficientes al sistema que no aporta ninguna información:

$$\frac{2}{3}b_{-2}+\frac{1}{24}b_{-1}+\frac{1}{24}b_1+\frac{2}{3}b_2=0$$

Pero que sin embargo sí que verifican nuestros coeficientes. Si en cambio intentamos obtener el orden cinco en la derivada el sistema también se vuelve incompatible, al aparecer la ecuación:

$$-\frac{4}{15}b_{-2}-\frac{1}{120}b_{-1}+\frac{1}{120}b_1+\frac{4}{15}b_2=0$$

Que vuelve al sistema incompatible. Es por eso que el orden buscado es 4.

- **Considerad $f=\cos(x)$. Construid un programa y calculad numéricamente $f'(x)$ en el intervalo $x\in[0,2\pi]$ utilizando las discretizaciones Δ_h, δ_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.

Cálculo de las derivadas forward y centrada

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

```
##EJERCICIO 2 E)
'''PARTE 1- CALCULO DE DERIVADAS
IMPORTAMOS LAS LIBRERIAS QUE NOS VAN A SER UTILES'''
import numpy
from numpy import *
import matplotlib.pyplot as plt
'''DEFINIMOS LA FUNCION EVALUAR EN LA FUNCION CON LA QUE VAMOS A TRABAJAR'''
def evaluar_coseno(t):
        f=cos(t)
return f
'''Y EN SU DERIVADA'''
def evaluar__menos_seno(t):
        f= -sin(t)
return f
'''AHORA SI, EXPONEMOS EL CODIGO DE LA DERIVADA FORWARD'''
def derivada_forward(t,f):
```

```
        '''INICIALIZAMOS UNA LISTA QUE SERA NUESTRA DERIVADA'''
        derivada=[]
        '''h ES EL SALTO DE UN VECTOR QUE SUPONEMOS EQUIESPACIADO '''
        h=t[1]-t[0]
        '''EXTENDEMOS NUESTRO VECTOR DE DERIVADAS, MIRAR .TEX PARA RAZONAMIENTO'''
        f=concatenate((f,array([f[1],f[2],f[3]])))
        t=concatenate((t,array([t[1],t[2],t[3]])))

        '''OBTENEMOS EL VECTOR DERIVADA MOVIENDONOS CON UN FOR'''
        for i in range (0,len(t)-3):
                derivada.append((-11/6*f[i]+3*f[i+1]-1.5*f[i+2]+1/3*f[i+3])/h)

        '''EXTRAEMOS EL VECTOR DERIVADA'''
        return derivada
def main():
        '''CREAMOS NUESTRO VECTOR TEMPORAL Y SU CORRESPONDIENTE IMAGEN'''
        t=linspace(0,2*pi,100)
        f=evaluar_coseno(t)
        '''CREAMOS UN VECTOR DE LA MISMA LONGITUD CON LA DERIVADA VERDADERA'''
        derivada_v=evaluar__menos_seno(t)
        '''EJECUTAMOS LA FUNCION DERIVADA FORWARD'''
        derivada=derivada_forward(t,f)
        '''PLOTEAMOS AMBAS FUNCIONES PARA COMPARARLAS'''
        plt.scatter(t,derivada,label='Derivada forward coseno',color='orange')
        plt.plot(t,derivada_v,label='Derivada verdadera coseno')
        plt.legend()
        plt.xlabel('t')
        plt.ylabel('$-\sin(t)$')
        plt.title('Derivada forward')
        plt.show()
if __name__ == '__main__':
        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:

$$f'(t+h) = f'(t+h-T), \qquad t, h \in \mathbb{R}, T > 0 \tag{8}$$

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 (8), nos damos cuenta de que los tres términos que nos faltan los podemos extraer de los tres primeros términos del vector (sin contar el primer elemento, pues es igual al último), 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:

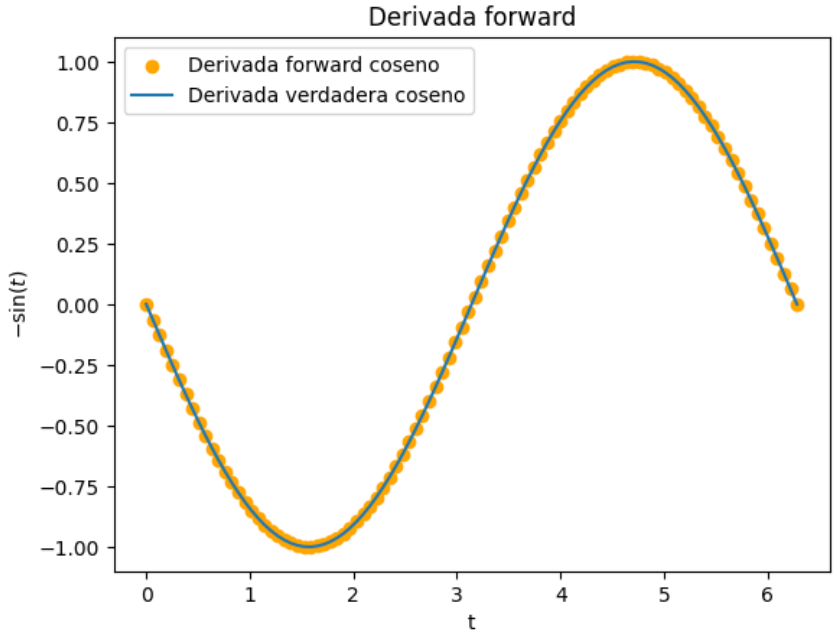


Imagen 1: Plot de la derivada forward frente a la derivada real

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

```
def derivada_centrada(t,f):
        derivada=[]
        h=t[1]-t[0]
        '''EXTENDEMOS NUESTROS VECTORES POR DELANTE Y DETRAS PARA PODER DERIVAR'''
        f=concatenate((array([f[-3],f[-2]]),f,array([f[1],f[2]])))
        t=concatenate((array([t[-3],t[-2]]),t,array([t[1],t[2]])))
        for i in range (2,len(t)-2):
                derivada.append((1/12*f[i-2]-2/3*f[i-1]+2/3*f[i+1]-1/12*f[i+2])/h)
        return derivada
def main():
        t=linspace(0,2*pi,100)
        f=evaluar_coseno(t)
        derivada_v=evaluar__menos_seno(t)
        derivada=derivada_centrada(t,f)
        plt.scatter(t,derivada,label='Derivada centrada coseno',color='orange')
        plt.plot(t,derivada_v,label='Derivada verdadera coseno')
        plt.legend()
        plt.xlabel('t')
        plt.ylabel('$-\sin(t)$')
        plt.title('Derivada centrada')
        plt.show()
if __name__ == '__main__':
        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 (8), 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:

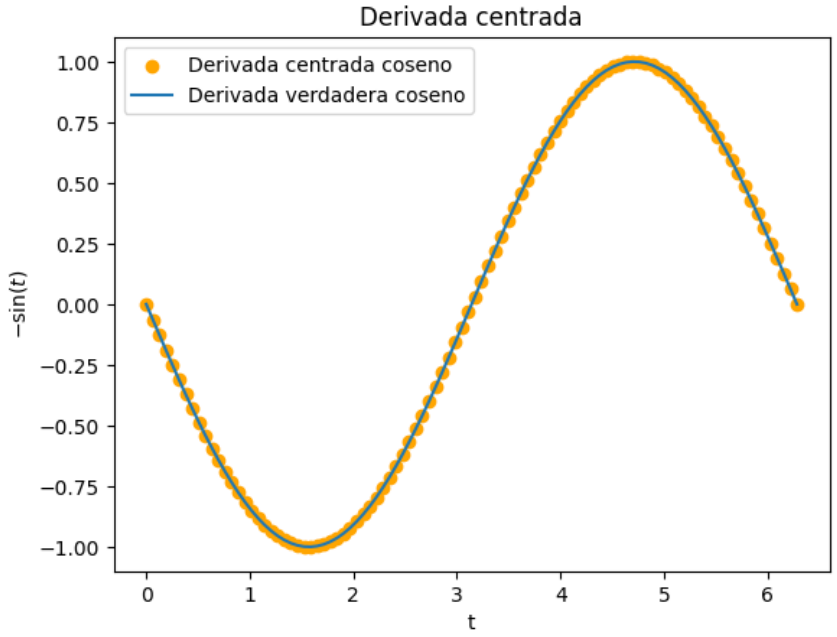


Imagen 2: Plot de la derivada centrada frente a la derivada real

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:

```
'''PARTE 2- CALCULO DE ERRORES'''
def calculo_errores(n_h):
        '''INICIALIZAMOS UNA LISTA DE ERRORES PARA CADA DERIVADA, Y UNA LISTA DE
            SALTOS h'''
        error_f=[]
        error_c=[]
        h=[]
        '''UTILIZAMOS UN VECTOR FOR PARA MOVERNOS EN EL VECTOR QUE MODIFICA h'''
        for i in range (0,len(n_h)):
                '''EN CADA ITERACION AUMENTA EL NUMERO DE TIEMPOS, Y DISMINUYE h'''
                t=linspace(0,2*pi,int(n_h[i]))
                '''METEMOS h EN LA LISTA EN CADA ITERACION'''
                h.append(t[1]-t[0])
                '''EJECUTAMOS LOS PROGRAMAS DE DERIVADAS'''
                f=evaluar_coseno(t)
                derivada_f=derivada_forward(t,f)
                derivada_c=derivada_centrada(t,f)
                derivada_v=evaluar__menos_seno(t)
                '''CALCULAMOS LOS ERRORES COMO LA MEDIA DEL ERROR ABSOLUTO, EN CADA h
                    '''
                error_f.append(mean(abs(derivada_f-derivada_v)))
                error_c.append(mean(abs(derivada_c-derivada_v)))
        '''PINTAMOS LOS ERRORES FRENTE A h'''
        plt.scatter(h,error_f,label='Error forward',color='red',s=7)
        plt.scatter(h,error_c,label='Error centrado',s=7)
        plt.legend()
        plt.xlabel('h')
        plt.ylabel('Error')
        plt.title('Errores frente a $h$')
```

```
        plt.show()
        '''PINTAMOS LOS ERRORES FRENTE A h, AMBOS EN ESCALA LOGARITMICA'''
        plt.scatter(log(h),log(error_f),label='Error logar tmico forward',color='red'
            ,s=7)
        plt.scatter(log(h),log(error_c),label='Error logar tmico centrado',s=7)
        plt.legend()
        plt.xlabel('$\log(h)$')
        plt.ylabel('$\log(Error)$')
        plt.title('Errores en escala logar tmica')
        plt.show()
        '''REALIZAMOS UN AJUSTE LINEAL DEL LOGARITMO TOMADO'''
        p_f=polyfit(log(h),log(error_f),1)
        p_c=polyfit(log(h),log(error_c),1)
        ''' LO REPRESENTAMOS FRENTE A LOS DATOS'''
        plt.scatter(log(h),log(error_f),color='red',s=5,label='Datos')
        plt.plot(log(h), log(h)*p_f[0]+p_f[1],label='Ajuste')
        plt.legend()
        plt.title('Ajuste error derivada forward')
        plt.xlabel('$\log(h)$')
        plt.ylabel('$\log(Error)$')
        plt.show()
        '''LO HACEMOS TAMBIEN CON LA DERIVADA CENTRADA '''
        plt.scatter(log(h),log(error_c),color='red',s=5,label='Datos')
        plt.plot(log(h), log(h)*p_c[0]+p_c[1],label='Ajuste')
        plt.legend()
        plt.title('Ajuste error derivada centrada')
        plt.xlabel('$\log(h)$')
        plt.ylabel('$\log(Error)$')
        plt.show()
return p_f,p_c

def main():
        '''CREAMOS EL VECTOR QUE NOS VA A DETERMINAR EL NUMERO DE ELEMENTOS DEL VECTOR
            TIEMPO,
        Y h COMO CONSECUENCIA'''
        n_h=linspace(10,1000,100)
        p_f,p_c=calculo_errores(n_h)
        print('Los par metros de la recta de ajuste de la derivada forward son '+str(
            p_f))
        print('Los par metros de la recta de ajuste de la derivada centrada son '+str
            (p_c))
if __name__ == '__main__':
        main()
```

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

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} \tag{9}$$

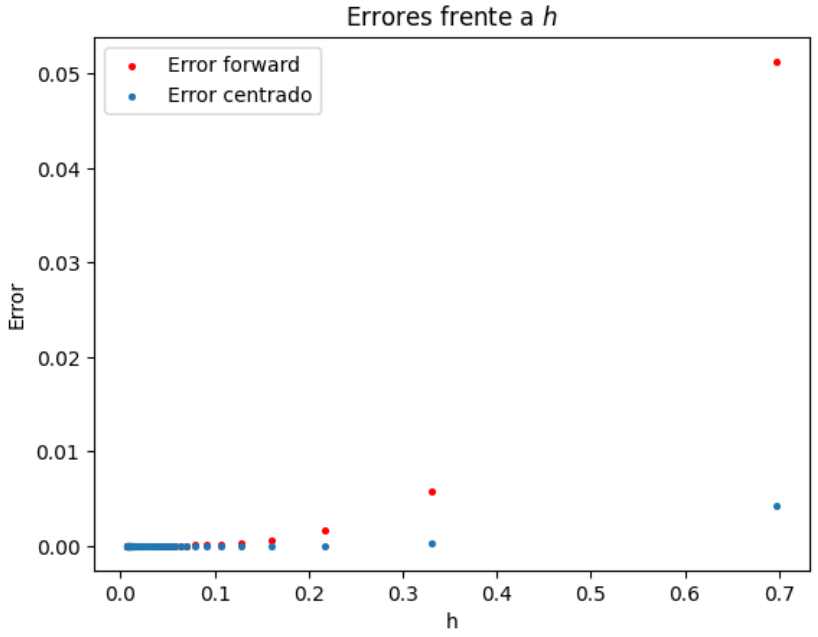
Esto implica particularmente, que:

$$f'(x) = \lim_{n\to \infty} \frac{f(x+\frac{1}{n}) - f(x)}{1/n} \tag{10}$$

La elección de las ecuaciones (9) y (10) 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 naturales, que serían como nuestro n en la expresión (10). Que la n aumente, implica que la h disminuya, y asintóticamente se obtiene lo que nos indican estas ecuaciones.[4]

[4]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 pricipio se ilustra de igual manera.

Con todo esto queremos llegar a que, al hacer n infinitamente grande, h se hará infintesimalmente 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 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. [5]
Los resultados obtenidos son:



[6] 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 $y = ax^p$, el logaritmo[7] 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:

[5] 2 Es por esto que no presentamos el valor numérico de los errores, pues no aportan más significado que el gráfico.
[6] 2 Los errores de las dos derivadas se solapan pues son muy similares.
[7] 2 A priori no importa la base elegida si lo que nos importa es la pendiente

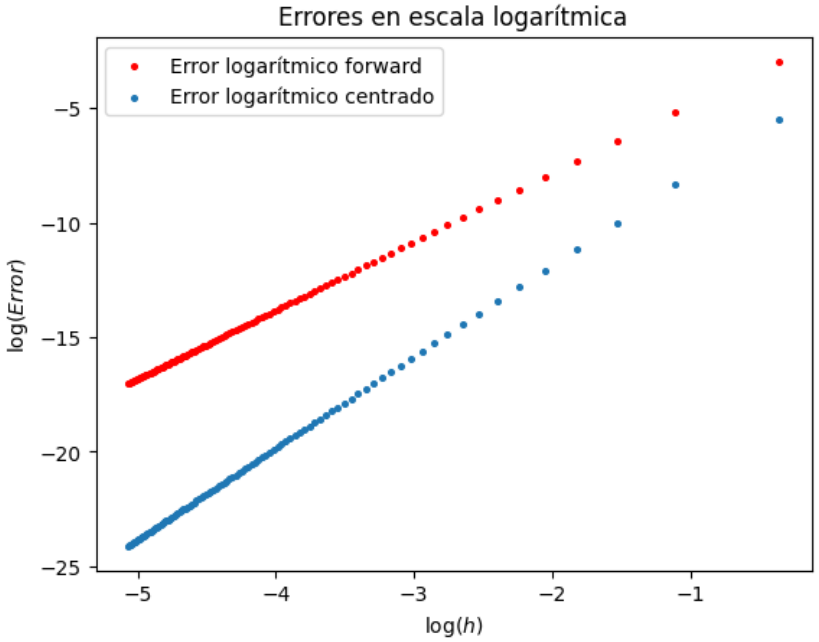


Imagen 4: Representación logarítmica del error frente al logaritmo de h

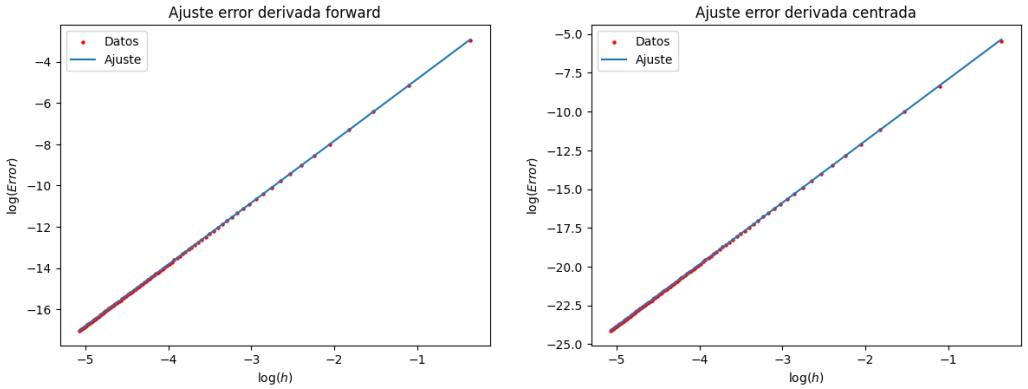
Además, por hipótesis se debería cumplir que:

$$(\log(h), \log(error(h))) \underset{h \longrightarrow 0}{\longrightarrow} (-\infty, -\infty)$$

Y podemos afirmar que se cumple. Para terminar este ejercicio vamos a calcular los parámetros asociados a los ajustes realizados anteriormente. Los datos obtenidos han sido:

```
Los parametros de la recta de ajuste de la derivada forward son [ 2.99958133
    -1.83817665]
Los parametros de la recta de ajuste de la derivada centrada son [ 3.98603836
    -3.91630893]
```

Es decir, que obtenemos un valor de $m_f \approx 3$ para la pendiente de la derivada forward y de $m_c \approx 4$ para la pendiente de la derivada centrada. Por lo que podemos afirmar que los resultados obtenidos son correctos (hemos obtenido el máximo orden de discretización).
Las gráficas asociadas a estos $polyfit$ son:



Imágenes 5 y 6: Polyfit derivadas forward y centrada

3. Ejercicio 3

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

α(grados)	0	30	60	90	120	150	180	210	240	270	300	330
δ(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 2π radianes (360º) 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:

```
        # APARTADO F

        '''DEFINIMOS LA FUNCI N QUE NOS VA A PERMITIR CALCULAR LOS COEFICIENTES DE
            FOURIER.
        SU  NICO  ARGUMENTO DE ENTRADA ES UN VECTOR DE IM GENES UNIFORMEMENTE
            DISTRIBUIDAS
        EN EL PERIODO.'''
        def coeficientes_fourier(delta):
        '''LA FUNCI N rFFT NOS DA LOS COEFICIENTES COMPLEJOS DE FOURIER'''
        c_i = fft.rfft(delta, norm='forward')
        '''LOS COEFICIENTES COSENOIDALES {a_i}, Y SINUSOIDALES {b_i} LOS OBTENEMOS
            MEDIANTE SIMPLES
        RELACIONES ALGEBRAICAS, USANDO QUE $c_i=\frac{a_i-ib_i}{2}$'''
        a_i=c_i+conj(c_i)
        b_i=(c_i-conj(c_i))*1j
        '''ELIMINAMOS b_i[0], PUES SABEMOS QUE c_i[0]=a_i[0/2], e.d. $c_i[0]\in\mathbb
            {R}$'''
        b_i=b_i[1:]
        '''HACEMOS QUE LA FUNCI N NOS DEVUELVA LOS COEFICIENTES'''
        return a_i,b_i,c_i
        def main():
        '''PRIMERO DEFINIMOS LOS VECTORES ALPHA Y DELTA QUE VAMOS A UTILIZAR'''
        alpha=linspace(0,330,12)
        delta=array([408,89,-66,10,338,807,1238,1511,1583,1462,1183,804])
        '''AHORA REPRESENTAMOS LOS PUNTOS OBTENIDOS CON LA FUNCI N SCATTER PARA
            POSTEIORMENTE
        COMPARARLO CON LOS RESULTADOS QUE OBTENGAMOS'''
        plt.scatter(alpha,delta,label='Datos  Bar n von Zach (enunciado)')
        plt.xlabel('Ascensi n recta (Grados)')
        plt.ylabel('Declinaci n (Minutos)')
        plt.legend()
        plt.title('Datos enunciado')
        plt.show()
        a_i, b_i, c_i = coeficientes_fourier(delta)
        '''AHORA EXOPONEMOS LOS COEFICIENTES DE FOURIER OBTENIDOS'''
        print('Los coeficientes complejos de la serie de Fourier son '+ str(c_i))
        print('Los coeficientes cosenoidales son '+str(a_i))
        print('Los coeficientes sinusoidales son '+str(b_i))
        return a_i,b_i,c_i
        if __name__ == '__main__':
        main()
```

Si ejecutamos el programa obtenemos:

```
Los coeficientes complejos de la serie de Fourier son [ 7.80583333e
    +02+0.00000000e+00j -2.05507183e+02+3.60113946e+02j
2.17083333e+01+1.08253175e+00j -2.16666667e+00-2.75000000e+00j
-5.41666667e-01+5.05181486e-01j  1.73850033e-01+1.36053580e-01j
8.33333333e-02+0.00000000e+00j]
Los coeficientes cosenoidales son [ 1.56116667e+03+0.j -4.11014367e+02+0.j
    4.34166667e+01+0.j
-4.33333333e+00+0.j -1.08333333e+00+0.j  3.47700065e-01+0.j
1.66666667e-01+0.j]
Los coeficientes sinusoidales son [-7.20227893e+02+0.j -2.16506351e+00+0.j
    5.50000000e+00+0.j
-1.01036297e+00+0.j -2.72107160e-01+0.j  0.00000000e+00+0.j]
```

Y la imagen:

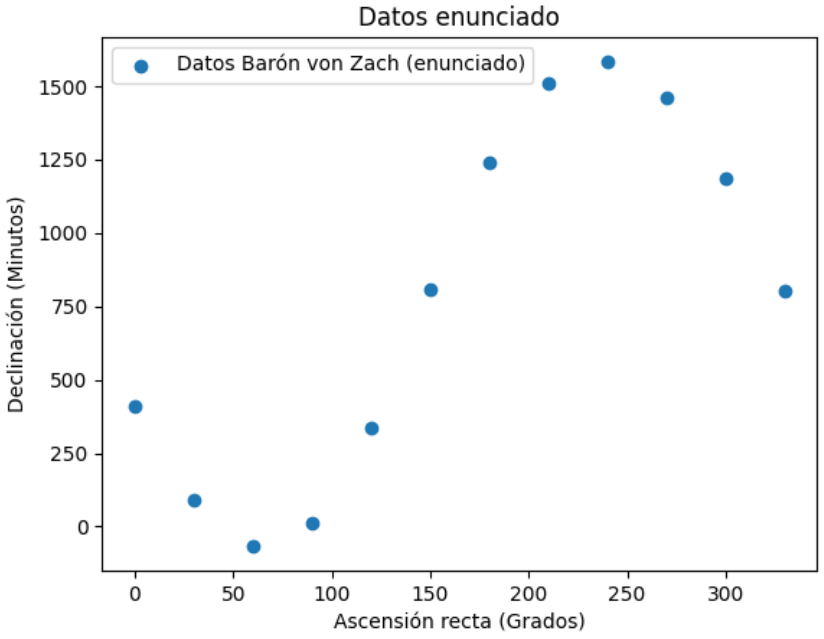


Imagen 7: Plot de los datos del enunciado.

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 :

$$c_k = c^*_{-k},\ k \geq 1$$

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:

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

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

Solución:

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

```
        # APARTADO G
        '''AHORA CONSIDERAMOS LOS COEFICIENTES DE FOURIER OBTENIDOS EN EL APARTADO
            ANTERIOR Y LOS REDONDEAMOS
        HASTA EL SEGUNDO TERMINO'''
        a_i,b_i,c_i=main()
        b_i=around(b_i,2)
        a_i=around(a_i,2)
        c_i=around(c_i,2)
        '''AHORA SIMPLEMENTE UTILIZAMOS LA DEFINICI N DE LA SERIE DE FOURIER Y LA
            EXPRESAMOS:'''
        print('Los coeficientes cosenoidales son :'+str(a_i))
        print('Los coeficientes sinusoidales son :'+str(b_i))
```

Y obtenemos por pantalla:

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

Y la serie de Fourier queda:

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

- **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:

```
        #APARTADO H
        '''UTILIZAMOS UNA FUNCION QUE TIENE COMO ARGUMENTOS DE ENTRADA
        UN VECTOR DE TIEMPOS (EN ESTE CASO GRADOS), LOS COEFICIENTES
        DE LA SERIE DE FOURIER OBTENIDOS ANTERIORMENTE, Y EL PERIODO
        DE LA FUNCION DE LA QUE TENEMOS QUE INTERPOLAR.'''
        def evaluar_delta(alpha,c_i,T):
        evalu=c_i[0]*ones(len(alpha))
        for i in range(1,len(c_i)):
        evalu=evalu+c_i[i]*exp(2*pi*1j*i*alpha/T)+conj(c_i[i])*exp(2*pi*1j*(-i)*alpha/
            T)
        return evalu
        def main():
        alpha=linspace(0,330,12)
        delta=array([408,89,-66,10,338,807,1238,1511,1583,1462,1183,804])
        alpha_largo=linspace(0,330,1000)
        T=360
        evalu=evaluar_delta(alpha_largo,c_i,T)
        '''PRINTEAMOS LOS DATOS DEL ENUNCIADO JUNTO CON LA INTERPOLACION
        CON MUCHOS DATOS PARA GARANTIZAR LA SUAVIDAD DE LA CURVA'''
        plt.plot(alpha_largo,evalu,label='Interpolaci n $\delta$')
```

```
        plt.scatter(alpha,delta,label='Datos enunciado')
        plt.xlabel('Alpha(grados)')
        plt.ylabel('Delta (minutos)')
        plt.legend()
        plt.title('Interpolaci n via Fourier')
        plt.show()
        if __name__ == '__main__':
        main()
```

Presentamos los resultados obtenidos:

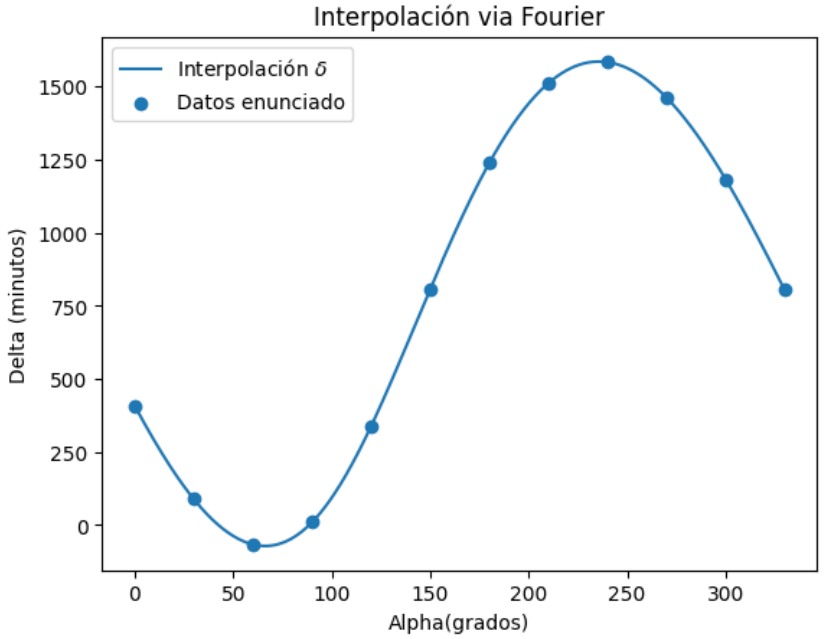


Imagen 8: Plot de los datos del enunciado frente a la interpolación.

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 2π periódica, como habíamos predicho.

- **Calcula la declinación que se espera a 45º de AR.**

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

```
        #APARTADO I
        '''PARA CALCULAR EL VALOR DE LA DECLINACION A 45   SIMPLEMENTE
        TENEMOS QUE UTILIZAR LA FUNCI N CREADA EN EL APARTADO
        ANTERIOR Y SUSTITUIR POR EL VALOR INDICADO'''
        def main():
        alpha_largo=linspace(0,330,1000)
        T=360
        evalu=evaluar_delta(alpha_largo,c_i,T)
        f_45=evaluar_delta(array([45]),c_i,T)
        '''PRINTEAMOS LOS DATOS DEL ENUNCIADO JUNTO CON LA INTERPOLACION
        CON MUCHOS DATOS PARA GARANTIZAR LA SUAVIDAD DE LA CURVA'''
        plt.plot(alpha_largo,evalu,label='Interpolaci n $\delta$')
        plt.scatter(45,f_45,label='Interpolaci n a 45  ')
        print('La interpolaci n de la funci n a 45 grados es '+str(f_45))
        plt.xlabel('Alpha(grados)')
```

```
        plt.ylabel('Delta (minutos)')
        plt.legend()
        plt.title('Interpolaci n a 45  ')
        plt.show()
        if __name__ == '__main__':
        main()
```

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

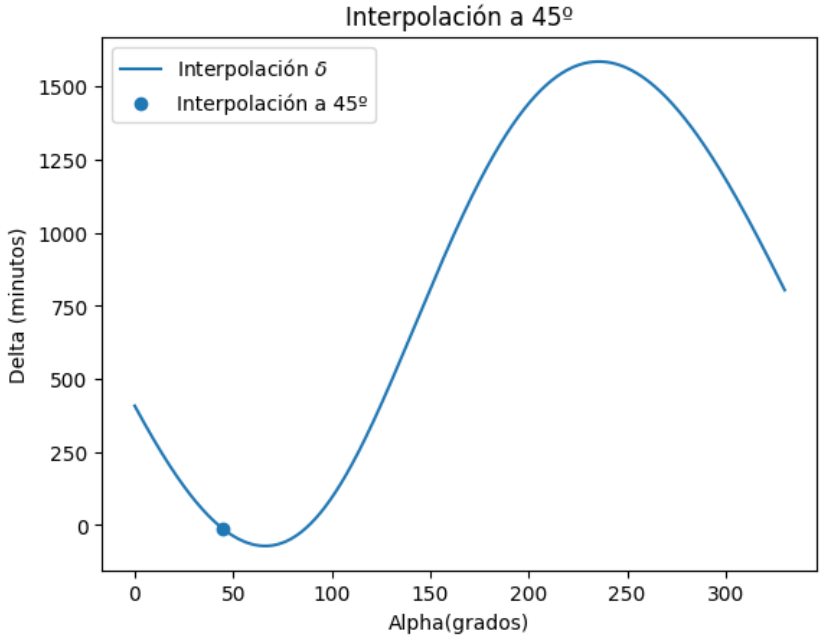


Imagen 9: Plot de los datos de la interpolación anterior frente al valor a 45º de ascensión recta.

- **Finalmente, utiliza la serie de Fourier estimada para estimar la tasa de cambio de δ en función de α 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:

```
        #APARTADO J
        '''CREAMOS LA FUNCION DERIVADA CON 3 ARGUMENTOS DE ENTRADA, EL VECTOR DE
            GRADOS, LOS COEFICIENTES
        DE FOURIER Y EL PERIODO DE LA FUNCION.'''
        def evaluar_delta_prima(alpha,c_i,T):
        '''CREAMOS UN VECTOR DE CEROS PARA INICIALIZAR EL VECTOR EVALUACION DE LA
            DERIVADA'''
        evalu=zeros(len(alpha))
        '''EMPEZAMOS EL BUCLE FOR EN EL PRIMER TERMINO PORQUE LA        DERIVADA DE
            LAS CONSTANTES ES NULA'''
        for i in range(1,len(c_i)):
        evalu=evalu+c_i[i]*(2*pi*1j*i/T)*exp(2*pi*1j*i*alpha/T)+conj(c_i[i])*(2*pi*1j
            *(-i)/T)*exp(2*pi*1j*(-i)*alpha/T)
        return evalu
        def main():
        '''CREAMOS UN VECTOR DE TIEMPOS QUE NOS PERMITA OBTENER UNA VERSION
            RAZONABLEMENTE SUAVE
        DE LA FUNCION'''
        alpha=linspace(0,330,1000)
```

```
        f_prima=evaluar_delta_prima(alpha,c_i,360)
        plt.plot(alpha,f_prima,label='Derivada en intervalo equiespaciado')
        f_prima_45=evaluar_delta_prima(array([45]),c_i,360)
        '''PRINTEAMOS EL VALOR DE LA FUNCION EN EL PUNTO DE INTERES'''
        print('El valor de la derivada en 45   es '+str(f_prima_45))
        plt.scatter([45],f_prima_45,label="f'(45)")
        plt.xlabel('Alpha(grados)')
        plt.ylabel('Derivada de $f$ (minutos/grados)')
        plt.legend()
        plt. title('Derivada e interpolaci n a 45  ')
        plt.show()
        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:

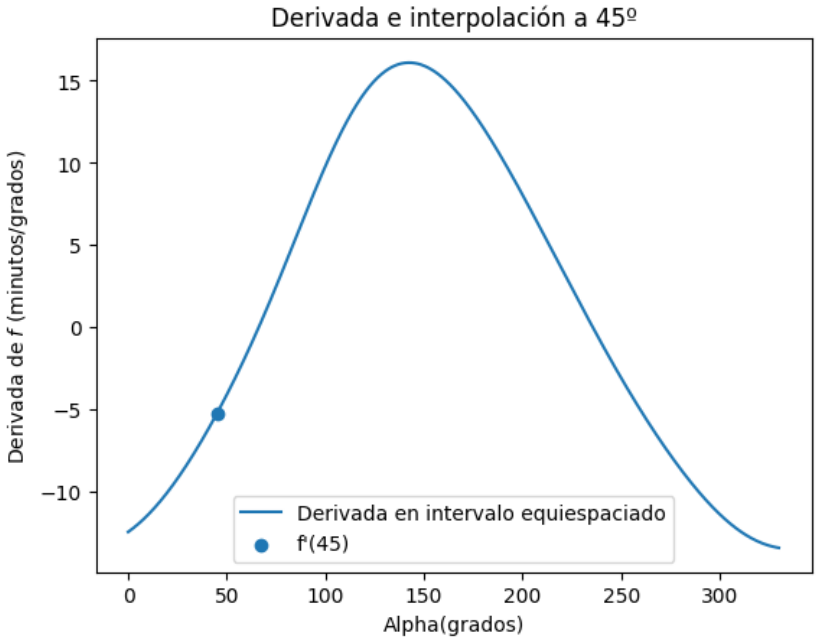


Imagen 10: Plot de los datos de la interpolación anterior frente al valor a 45º de ascensión recta.

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.