TEC | Tecnológico de Costa Rica

Área Académica de Ingeniería Física

Física Computacional I

Tarea 1

Estudiante:
Ever Ortega Calderón

Fecha de entrega:
4 de marzo de 2021

Profesores:
Álvaro Amador Jara
Jose Esteban Pérez Hidalgo

I Semestre 2021

1)

a) Pseudocódigo para regla integral de Newton-Cotes, en este caso Simpson:

Se deberá importar a las bibliotecas numpy (para crear los puntos en el intervalo respectivo) y sympy (para trabajar con variables simbólicas).

Se deberá preguntar todos los datos del problema al usuario, esto podría generar errores de no meter los datos como están esperados, por ende, revisar con try y except que no existan errores de tipo de dato, además de usar un ciclo, para que si el usuario se equivoca pueda volver a ingresar los datos.

Definir la función "crear_funcion", la cual será una línea recta, por ende usar los datos que usó el usuario para calcular la pendiente y la intersección con el eje de las abscisas, que la función me devuelva la ecuación de la recta con la x de forma simbólica.

Se debe definir el intervalo [a,b] que abarca la integral, es decir, los límites de la integral, que irán relacionados a el tiempo inicial y final que ingresó el usuario, si están al revés la integral puede dar negativa, por ende revisar con un if si el a es menor que el b, así como la cantidad de puntos deseados y considera que el N debe ser impar para que el método funcione, igualmente hacer las excepciones si el usuario mete un tipo de dato que no es el esperado.

Calcular el valor h que corresponde al espaciamiento como (b-a)/(N-1)

Crear el arreglo "preimágenes" de puntos de x, que consiste en a, a+h, a+2h… hasta llegar a b, esto con la función linspace de numpy.

Crear un arreglo "imágenes" que serían las imágenes del arreglo "preimágenes" bajo la función, esto con un for y usando la función "creacion_funcion"

Emplear h y el arreglo "imágenes", además de la fórmula de Simpson para calcular el valor de la integral respectiva:

$$= \frac{h}{3}\left(f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \cdots + f(b)\right)$$

Error: $O(h^4)$

Mostrar el resultado de la integral.

b) Códigos generados se encuentran en: https://github.com/ever2706/FCI-Tarea1.git

c)

Cantidad de puntos	Resultado que genera
5	75.0

7	75.00000000000001
9	75.0
11	75.0
13	75.00000000000001

Se tiene un caso de MRUA, empleando la ecuación cinemática:

$$x = \left(\frac{v_f + v_0}{2}\right) * t$$

Con los datos respectivos, se obtiene que x=75 m, así que con una incertidumbre de 1% el dato a encontrar mediante la aproximación sería x=(75 ± 0.75)m se observa que desde que se hace la primera iteración con 5 puntos se ve que el resultado es sumamente parecido al dato exacto, esto se puede deber a que el programa muestra una cantidad de ceros y después algún número extra, como en 13 puntos, por ende lo toma como 75.0.

Además, se ve que el método converge sumamente rápido, esto se puede deber a la simpleza de la función que se trató, pues se habla de una recta cuyo incremento en las ordenadas es de tan solo 0.5 en el intervalo de análisis, por lo cual el método a pesar de ser una aproximación al valor real consigue llegar a este de forma o al menos a una aproximación exacta muy rápida.

2)

a) Pseudocódigo para Newton-Raphson:

Definir la función F a trabajar mediante una función que devuelva la función F, esto ayudará a la hora de crear las imágenes y la derivada.

Solicitar la aproximación inicial x0 deseada, así como el valor de h que se desea para el cálculo de la integral numérica, revisar que el tipo de dato ingresado sea el adecuado.

El procedimiento se debe repetir una cantidad determinada de veces, por ende darle la opción al usuario de ver qué cantidad de iteraciones desea, revisar que efectivamente ingrese un número natural.

En un ciclo que se repita la cantidad de iteraciones ingresadas: calcular la imagen de x0 bajo f, la imagen de x0 bajo la derivada de F, para esto usar la definición de la derivada numérica centrada, calcular la nueva aproximación como x0-(F(x0)/ F’(x0)) y definir este cálculo como la nueva raíz.

Mostrar al usuario la aproximación de la raíz.

b) Códigos generados se encuentran en: https://github.com/ever2706/FCI-Tarea1.git

c)

La función por trabajar es:

$$x = 0.005 * t^2 - 5$$

La cual es una cuadrática que al hacer su x=0 y resolver la ecuación, la solución positiva (pues buscamos un valor de tiempo y el resultado negativo no tendría sentido físico) es 31.6227766, con lo cual una incertidumbre el 1% representa (31.6227766±0.316227766) s

Para llegar a este valor con un h=0.0001 y una aproximación inicial de 1, se requieren más de 5 iteraciones, pues con estos valores da una raíz de 41.24542607136558, mientras que con 6 iteraciones da 31.64201586868082.

Sumamente importante considerar el valor de h que se asigna para el respectivo cálculo de la derivada y la aproximación inicial, pues esto varía el resultado que la máquina brinda, a menor h y mejor la aproximación con respecto al valor real, el método convergerá de forma más rápida.

3)

¿Cuáles son las complicaciones que considera que tiene la preparación de cada código?

La mayor complicación que considero es la comprensión de los métodos a aplicar, es decir, la buena aplicación de la fórmula de Simpson en el código, asimismo como la correcta ejecución de Newton-Raphson, pues si se entiende bien esto, entonces programarlo será seguir el algoritmo.

También la comprensión del fenómeno físico a programar podría ser la segunda complicación, debido a que si esto no se tiene con claridad entonces el código no sería capaz de cumplir los objetivos a cabalidad, ni de darle la correcta interpretación a los resultados.

Y por último el manejo de excepciones, pues al intervenir con cálculos numéricos pueden existir procedimientos que den error si no se tienen en cuenta dichos errores o bien que se ingrese un dato incorrecto que altere los resultados.

¿Cuán difícil es lograr la precisión indicada?

Para la integración se hace sumamente rápido, pues como se mencionó antes en este documento la forma de la función y la eficiencia del método logran que con pocos puntos se llegue a un valor aceptado de forma rápida.

Para el proceso de la raíz el método converge muy rápido con los parámetros indicados anteriormente en este texto, pero un poco más lento que como se evidencio en el caso de la integral, a la sexta iteración ya se consigue una incertidumbre menor al 1%.

¿Existen funciones en bibliotecas de Python que permitirían realizar los cálculos solicitados? ¿Cuáles?

Sí existen.

La librería de SciPy ofrece la función quad para integración y más específicamente en las reglas de Newton Cotes ofrece simps que ejecuta la regla se Simpson.

Para Newton- Raphson tambien SciPy ofrece optimimize el cual ejecuta newton, es decir optimize.newton y usa el método de Newton Raphson para encontrar un cero, es decir una raíz.