Examen Final

Ramírez David R.

Department: Computación
Course: Pensamiento Computacional 108
Instructor: Benito Granados-Rojas, PhD.
Date: November 15, 2022

Tecnológico de Monterrey

1 Introduction

En este trabajo se emplearan 3 algoritmos computacionales en Python para resolver tres distintos problemas. Se aplicará Simpson $3/8$ Compuesto para una integral definida, se aplicará Runge-Kutta de órden 4 para una ecuación diferencial ordinaria de valor inicial, y se aplicará el algoritmo de la ruta más corta para una red.

1.1 Simpson $3/8$ Compuesto

El Método de Simpson $3/8$ Compuesto es un método de integración numérica para integrales definidas. Este método es una derivación del la regla de Simpson 3/8 Simple, donde se propone una interpolación cúbica para aproximar la integral. Para el caso compuesto, se divide el intervalo en n subsecciones para aplicar el método simple en cada una de estas (Wikipedia 2022c). De esta forma, el método se describe en (1). Sin embargo, es más fácil codificar (2).

$$\int_a^b f(x) \approx \frac{3}{8}h[f(x_0) + 3\sum_{i\neq 3k}^{n-1} f(x_i) + 2\sum_{j=1}^{n/3-1} f(x_{3j}) + f(x_n)] \tag{1}$$

$$\int_a^b f(x) \approx \frac{3}{8}h[f(x_0) + 3\sum_{i=1}^{n-1} f(x_i) - \sum_{j=1}^{n/3-1} f(x_{3j}) + f(x_n)] \tag{2}$$

Para ambos casos $h = \frac{b-a}{n}$

1.2 Runge-Kutta de Orden 4

El método de Runge-Kutta (RK) es un método para resolver ecuaciónes diferenciales ordinarias de manera numérica. Este método es aplicado particularmente para el "problema del valor inicial" o ecuaciones diferenciales cuyo valor inicial es conocido, partiendo de la ecuación y el valor inicial conocido (3), el método se aplica de manera iterativa hasta conocer el valor de la variable dependiente en el punto deseado (Wikipedia 2022b).

$$y' = f(x, y),\ y(x_0) = y_0 \tag{3}$$

El método RK es una familia de métodos, en particular, RK de orden 4 o RK4 se comporta de la siguiente manera:

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4}$$

Donde:

$$\begin{cases} k_1 = & f(x_i, y_i) \\ k_2 = & f(x_i + h/2, y_i + k_1 h/2) \\ k_3 = & f(x_i + h/2, y_i + k_2 h/2) \\ k_4 = & f(x_i + h, y_i + k_3 h) \\ x_i = & x_0 hi \end{cases}$$

1.3 Ruta más corta

El algoritmo de la ruta más corta es un método para determinar la ruta más corta en una red partiendo de un nodo origen hasta un nodo final. La ruta más corta en una red será aquella cuya suma de tiempos de nodos por los que pasará sea la menor. El algoritmo está diseñado para determinar la ruta más corta desde el origen hasta cada uno de los nodos, de esta manera, en cada iteración se avanza hacia el nodo final determinando las rutas más cortas en cada iteración. Al finalizar el algoritmo se habrá determinado la ruta más corta y se regresará el tiempo (Wikipedia 2022a).

2 Procedures and Results

2.1 Simpson $3/8$ Compuesto

Para esta sección, se empleó el método Simpson $3/8$ Compuesto para resolver la ecuación (5). El único módulo importado fue `numpy` para calcular el seno y

obtener π.

$$\int_{\frac{1}{2\pi}}^{2} \sin(1/x)dx = 1.11407449 \tag{5}$$

Primeramente, se planteó la función a integrar como una función externa.

Listing 1: Codificación de función

```
def f(x):
    f = np.sin(1/x)
    return f
```

Después, ya en la función principal, se definierón los parámetros iniciales a, b, n, h y la variable `integral` que almacenará los resultados de cada sumatoría. Cabe destacar que el número de subdivisones n que se encontró suficiente para obtener el resultado esperado fue $n = 900$.

Listing 2: Parámetros iniciales

```
a = 1/(2*np.pi)
b = 2
n = 3*3000
h = (b-a)/n
integral = 0
```

Posteriormente, se aplica el método de la siguiente forma:

$$\begin{aligned} integral &= 3\sum_{i=1}^{n-1} f(x_i) \\ integral &= integral - \sum_{j=1}^{n/3-1} f(x_{3j}) \\ integral &= integral + f(x_0) + f(x_n) \\ integral &= integral * \tfrac{3}{8}h \end{aligned}$$

Lo anterior, se codifica con el siguiente segmento de código:

Listing 3: Ejecución del método

```
for i in range(1,n):
    integral += f(a+h*i)
integral *= 3
for i in range(1,n//3):
```

```
        integral -= f(a+3*i*h)
    integral += f(a)+f(b)
    integral = 3*h/8*integral
    print("integral = ",integral)
```

Con lo anterior, se obtuvo:

$$\int_{\frac{1}{2\pi}}^{2} \sin(1/x)dx = 1.1140744942675347$$

2.2 Runge-Kutta de Orden 4

Para esta sección, se aplicó el método RK4 para resolver la ecuación (6), en el punto $x = 1.4$ con valor inicial $y(0) = 0$ y $h = 0.1$. En este caso, ningún módulo externo fue necesitado.

$$y' = 1 + y^2 \approx 5.7919748 \tag{6}$$

Al igual que la sección anterior, primero se codificó la función a resolver como una función externa.

Listing 4: Codificación de función

```
def g(x,y):
    g = 1+y**2
    return g
```

Ya en la función principal, se declaraón los parámetros iniciales $h, x_f, x = x_0, y = y_0$.

Listing 5: Parámetros iniciales

```
h = 0.1
xf = 1.4
x = 0
y = 0
```

Finalmente, para realizar el método RK4, dentro de un `while` que continuará mientras $x_i \leq x_f$, se ejecurá iterativamente el método como se muestra en (4). Al salir de ciclo, solo se imprimirá y.

Listing 6: Parámetros iniciales

```
while x<=xf:
    k1 = g(x,y)
    k2 = g(x+h/2,y+k1/2*h)
    k3 = g(x+h/2,y+k2/2*h)
    k4 = g(x+h,y+k3*h)
    y += h/6*(k1+2*k2+2*k3+k4)
    x += h
print(y)
```

Después de ejecutar el código anterior, se obtuvo:

$$y' = 1 + y^2 = 5.7919748000064037$$

2.3 Ruta más corta

En está sección, se aplicó el algoritmo de la ruta más corta para la red mostrada en Fig. 1 desde el nodo 1 hasta el nodo 8. En esta implementación tampoco fue necesario ningún módulo externo.

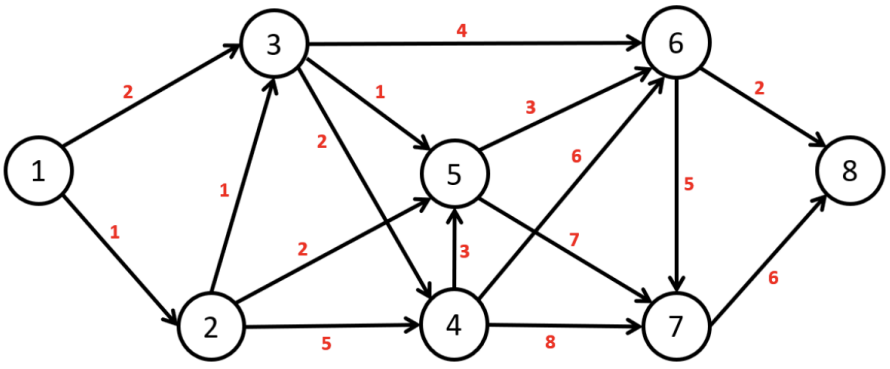

Figure 1: Red a procesar

.

Primeramente, se declararon los parámetros iniciales, `nodes` y `M` (Matriz de

conexiones). La constante `nodes` es un `string` que guarda el nombre de los nodos, la constante `M` es una lista de listas que contiene la distancia que hay entre un nodo y otro, cada elemento M_{ij} es el costo o la distancia que hay entre el nodo i y el nodo j, para facilitar la codificación de esta matriz, se empleó la constante auxiliar `x` para los casos en los que no hay un paso directo de un nodo a otro.

Listing 7: Parámetros iniciales

```
nodes='12345678'
x = '-'
    #1 2 3 4 5 6 7 8
M= [[x,1,2,x,x,x,x,x],\
    [x,x,1,5,2,x,x,x],\
    [x,x,x,2,1,4,x,x],\
    [x,x,x,x,3,6,8,x],\
    [x,x,x,x,x,3,7,x],\
    [x,x,x,x,x,x,5,2],\
    [x,x,x,x,x,x,x,6],\
    [x,x,x,x,x,x,x,x]]
```

A continuación, se puede ejecutar el algoritmo de la ruta más corta. Para ello, primero se declaran dos variables auxiliares: `menor` que es una lista de listas que contendrá la distancia más corta para llegar a un nodo i y el nodo inmediato anterior por la que esta ruta más corta (para el nodo i) tendrá que pasar, y `dists` que es una lista que contendrá el tamaño de las distancias más cortas a cada nodo desde el nodo 1.

Listing 8: Variables Auxiliares

```
menor = [[0,0,0]]*len(M)
dists = [0]*len(M)
```

Posteriormente, se analizará, en un ciclo `for`, columna por columna de la matriz de conexiones para determinar qué ruta es la más optima para llegar a ese nodo. Para ello, dentro del ciclo, se empleará un variable auxiliar temporal `pos` que contedrá las posibles rutas y sus distancias para llegar al nodo i. Seguidamente, en un ciclo anidado, se recorrerá el vector `dists` para cada nodo que pueda

llegar a el nodo i y se guardará la distancia final de la ruta en la variable `pos`. Al salir del ciclo anidado, se determinará la ruta más corta dentro de `pos` y se añadirá esa ruta a la lista `menor`, también se guardará cual fue la distancia más corta para el nodo i en la variable `dists[i]`.

Listing 9: Ruta más corta

```
for j in range(1,len(M)):
    pos = []
    for i in range(0,len(M)):
        if M[i][j] != x:
            dis = dists[i] + M[i][j]
            pos.append([dis,nodes[i],nodes[j]])
    menor[j] = min(pos)
    dists[j] = menor[j][0]
```

Al finalizar este ciclo, se habrá obtenido una lista que contiene las distancias más cortas a cada nodo desde el nodo 1. También, se habrá obtenido la ruta más corta a cada nodo. Por lo tanto, solo falta determinar cuáles nodos habría que seguir para llegar al nodo final 8. Para ello, se declara una variable `s` que contedrá los nombres de los nodos de la ruta, cada nodo de la ruta seguirá el siguiente formato:

Listing 10: Nodos en la ruta

```
s[i] = menor[s[i+1]-1,1]
```

Mediante un ciclo que continuará hasta que el nodo sea 1, se recorrá el vector `menor` y se añadirá a la ruta `s` el segundo elemento de `menor[i]` y este elemento dictará cuál es el nodo que sigue en `menor`. Cabe aclarar que, esto solo es posible porque los nombres de los nodos son números que tienen relación con el orden de la lista `menor`.

Listing 11: Ruta final

```
s='8'
let = menor[-1]
while let[1]!='1':
  let = menor[int(let[1])-1]
  s = (l[2]) +'-'+ s
```

```
        s = '1-'+s
```

Al finalizar, solo basta con imprimir los resultados:

Listing 12: Resultados

```
Nodos de la ruta mas corta:  1-3-6-8
Distancia total:  8 unidades
```

3 Conclusions

Como se puede apreciar, los métodos computacionales sirven para resolver problemas cuya solución analíticas es muy compleja o, incluso, imposible de obtener. Es interesante saber que hoy en día existen una gran variedad que métodos que permiten obtener resultados muy precisos, y adecuar estos métodos al contexto no es muy complicado, sin embargo, es necesario comprender cómo funciona el método para poder modificarlo. Por otro lado, también es importante la variedad de problemas en las que estos algoritmos son aplicables. Finalmente, hoy más que nunca resulta necesario contar con una visión general de los métodos numéricos y algoritmos que existen para resolver determinados problemas, pues emplearlos correctamente puede ahorrar mucho tiempo y se pueden obtener muy buenos resultados.

4 Repository

https://github.com/A01338802/ComputacionAplicada_EF.git

References

[1] Wikipedia. *Algoritmo de Dijkstra*. 2022. URL: https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra.

[2] Wikipedia. *Método de Runge-Kutta*. 2022. URL: https://es.wikipedia.org/wiki/M%C3%A9todo_de_Runge-Kutta.

[3] Wikipedia. *Regla de Simpson*. 2022. URL: https://es.wikipedia.org/wiki/Regla_de_Simpson.