Métodos numéricos - Tarea 4
Interpolación lineal

Salim Vargas Hernández

9 de septiembre del 2018

1. Introducción

La interpolación es un método que busca predecir o estimar nuevos puntos a partir de un conjunto dado de puntos que provienen de la misma fuente. La interpolación busca encontrar una *función interpolante* $f(x)$ tal que pase por todos los puntos dados (x_i, y_i), i. e. $f(x_k) = y_k$ para $k = 1, \ldots, n$ donde n es el número de puntos dados.
En el caso de la ***interpolación lineal***, la función interpolante es de la forma $f(x) = a_0 + a_1 x$, es decir, segmentos de recta entre cada dos puntos.

2. Desarrollo

2.1. Interpolación lineal

Dados n puntos (x_i, y_i) se trazan segmentos de recta que los unen, cada uno de estos segmentos está dado por:

$$f(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i)$$

Donde $f(x_i) = y_i$ y $f(x_{i+1}) = y_{i+1}$.
Cada vez que se quiera aproximar un nuevo valor x, deberá localizarse el intervalo $[x_i, x_{i+1}]$ al que pertenece y usar estos valores, junto con sus respectivos valores de y para, obtener la función interpolante y posteriormente evaluarla en x.

2.2. Búsqueda de índices de valores

Para aproximar el valor de un nuevo dato mediante interpolación, hace falta encontrar el intervalo en el que se encuentra, para ello se necesita realizar una búsqueda sobre el vector de x_i.

Búsqueda binaria

Dado un vector ordenado, se busca el par de elementos entre los cuales se encuentra un determinado valor. Este algoritmo va bisectando el intervalo hasta encontrar el valor deseado, encontrando el intervalo deseado en un tiempo logarítmico en el peor de los casos ($O(log_n)$). El **algoritmo 1** muestra el pseudocódigo de la *búsqueda binaria*.

Hunting search

A partir de una aproximación o estimación de la posición del intervalo en que que se encuentra un valor, *hunting search* busca en las vecindades de esta hasta que encuentra un intervalo en el que está contenido el valor, para luego hacer búsqueda binaria dentro del nuevo intervalo. El **algoritmo 2** muestra el pseudocódigo de *hunting search*.
Se debe cuidar que el algoritmo no salga de los límites del vector al duplicar el paso, para evitar hacer referencia a índices no existentes.

Algoritmo 1 Búsqueda binaria

Entrada:
Vector ordenado *vector* = $[x_0, x_1, \ldots, x_{n-1}]$
Valor a buscar *valor*
índiceIzquierdo e *índiceDerecho* para comenzar la búsqueda
Salida: *índiceIzquierdo* del intervalo donde se encuentra el *valor* buscado

Si *valor* está dentro de los límites del intervalo marcado por *índiceIzquierdo* e *índiceDerecho* **entonces**
  Mientras *índiceDerecho* − *índiceIzquierdo* > 1 **hacer**
    $\textit{índiceMedio} = \textit{índiceIzquierdo} + \frac{\textit{índiceDerecho} - \textit{índiceIzquierdo}}{2}$
    Si *valor* > *vector*[*índiceMedio*] **entonces**
      índiceIzquierdo = *índiceMedio*
    Si no
      índiceDerecho = *índiceMedio*
    Fin Si
  Fin Mientras
  Devolver *índiceIzquierdo*
Si no
  Devolver *índiceIzquierdo* del primer o último intervalo según corresponda
Fin Si

Algoritmo 2 Hunting search

Entrada:
Vector ordenado *vector* = $[x_0, x_1, \ldots, x_{n-1}]$
Valor a buscar *valor*
índiceAproximado para comenzar la búsqueda
Salida: *índiceIzquierdo* del intervalo donde se encuentra el *valor* buscado

paso = 1
Si *valor* está dentro de los límites del *vector* **entonces**
  Si *valor* < *vector*[*índiceAproximado*] **entonces**
    Mientras *valor* < *vector*[*índiceAproximado* − *paso*] **hacer**
      índiceAproximado = *índiceAproximado* - paso
      Duplicar el *paso*
    Fin Mientras
    Devolver búsqueda_binaria(*vector*, *valor*, *índiceAproximado-paso*, *índiceAproximado*)
  Si no
    Mientras *valor* > *vector*[*índiceAproximado* + *paso*] **hacer**
      índiceAproximado = *índiceAproximado* + paso
      Duplicar el *paso*
    Fin Mientras
    Devolver búsqueda_binaria(*vector*, *valor*, *índiceAproximado*, *índiceAproximado*+*paso*)
  Fin Si
Si no
  Devolver *índiceIzquierdo* del primer o último intervalo según corresponda
Fin Si

3. Resultados

Se tomaron los datos del archivo `normal_data.csv`. Se usaron los datos de las primeras dos columnas para realizar interpolación lineal y las búsquedas. El vector de x_i corre desde 0 hasta 4 en intervalos de 0.1.
El archivo `normal_data.csv` contiene además los datos de y para valores en intervalos de 0.01; después de realizar los cálculos de interpolación lineal para intervalos de 0.1, se usaron estos datos para calcular el error al predecir valores en intervalos de 0.01.
El **error absoluto promedio** obtenido fue de 8.10705×10^{-5}.

Se desarrolló un programa que permite al usuario estimar por interpolación lineal, los valores para y dada una x ingresada por consola. El programa permite al usuario realizar varias consultas del valor aproximado de y, el primer valor ingresado para x se busca usando *búsqueda lineal* y los siguientes usando *hunting search*, partiendo del índice anterior como aproximación.
Si el valor requerido por el usuario está fuera de los límites ($[0,4]$), se extrapola el valor para y usando la función resultante de interpolar en el primer o último intervalo según corresponda.
La figura 1 muestra un ejemplo de la salida en consola del programa.

```
Ingrese un valor entre 0 y 4: 3.14159

El valor aproximado para x = 3.14159 es ŷ = 0.499146

¿Aproximar un nuevo valor? Y/n : Y

Ingrese un valor entre 0 y 4: 1.19

El valor aproximado para x = 1.19 es ŷ = 0.38287

¿Aproximar un nuevo valor? Y/n : Y

Ingrese un valor entre 0 y 4: 5.34
        CUIDADO. Valor fuera del intervalo, se usará extrapolación

El valor aproximado para x = 5.34 es ŷ = 0.500238

¿Aproximar un nuevo valor? Y/n : Y

Ingrese un valor entre 0 y 4: -3.5
        CUIDADO. Valor fuera del intervalo, se usará extrapolación

El valor aproximado para x = -3.5 es ŷ = -1.39405

¿Aproximar un nuevo valor? Y/n : n
```

Figura 1: Ejemplo de salida del programa

4. Conclusiones

La interpolación lineal resultó ser lo suficientemente certera para aproximar los datos nuevos, obteniendo un error promedio de 8.10705×10^{-5}.
Aplicar *hunting search* al realizar búsquedas subsecuentes, evita pasos innnecesarios de *búsqueda lineal*, en especial para arreglos muy grandes, aunque tal vez este ejercicio no lo ejemplifica plenamente.