Métodos Numéricos
Trabajo Práctico Final
Método de búsqueda **k**-vector

Dores Piuma Francisco
Moser Plugge Tylbert
Noblía Martín
Vieras Fabián

Profesor:

Diego Passarella

Universidad Nacional de Quilmes

2013

1 Resumen

El método de búsqueda **k**-vector es un algoritmo que permite realizar búsquedas en bases de datos estáticas, con una mejora sustancial con respecto a otros métodos clásicos como el método binario, además el costo computacional del mismo es independiente del tamaño de la base de datos. Este método nació como una necesidad en el campo aeroespacial([1]), donde a través del método se resuelve con éxito el problema conocido como *star-tracker*[1]. Pero este método al ser su formulación general a cualquier base de datos(estática o pseudoestática) puede ser aplicado a una amplia gama de problemas, como:

- Interpolación (1D o 2D *look-up tables*)
- Búsqueda de ceros de funciones no-lineales
- Inversión de funciones no-lineales
- *Sampling* de distribuciones de probabilidades

En el presente informe vamos a repasar cuales son sus formulaciones matemáticas y además vamos a presentar dos implementaciones para mostrar el rango de aplicación del método.

2 Desarrollo

El problema de búsqueda en un rango de valores de una base de datos estática $\mathbf{y}(n)$ con un número de valores $n \gg 1$ es encontrar el subconjunto de elementos que satisfacen $\mathbf{y}(\mathbf{I})$, donde $\mathbf{I}$ es un vector de índices tal que $\mathbf{y}(\mathbf{I}) \in [y_a, y_b]$ donde $y_a < y_b$. Este problema se resuelve habitualmente con el método de búsqueda binario, el cual tiene un orden de complejidad $\mathcal{O}(2log_2(n))$ El método de búsqueda **k**-vector tiene en cambio un orden de complejidad invariante con respecto a n y es de $\mathcal{O}(3)$.

La metodología para implementar este método es la siguiente:

Sea $\mathbf{y}(n)$ un vector de datos de dimensión $n \gg 1$ y sea $\mathbf{s}(n)$ el mismo vector pero ordenado de manera ascendente, osea $\mathbf{s}(i) \leq \mathbf{s}(i+1)$, luego sea $\mathbf{s}(i) = \mathbf{y}(\mathbf{I}(i))$, donde $\mathbf{I}$ es un vector de índices que posee la información del ordenamiento. Definimos a:

$$y_{max} = \max_i\{\mathbf{y}(i)\} = \mathbf{s}(n) \tag{1}$$

$$y_{min} = \min_i\{\mathbf{y}(i)\} = \mathbf{s}(1) \tag{2}$$

Luego el corazón del algoritmo es el armado de una recta que conecta los puntos: $[1, y_{min} - \delta\epsilon]$ con $[n, y_{max} + \delta\epsilon]$, donde :

$$\delta\epsilon = (n-1)\epsilon \tag{3}$$

y ϵ es la precisión relativa de una computadora común ($\epsilon \approx 2{,}22 \times 10^{-16}$) con este pequeño paso aseguramos que $\mathbf{k}(1) = 0$ y que $\mathbf{k}(n) = n$ y asi se simplica la implementación ahorrando muchos comparaciones. Podemos escribir la ecuación de la recta como:

$$z(x) = mx + q \tag{4}$$

[1]En la actualidad se utilizan otros métodos más avanzados, pero el corazón de los mismos sigue siendo **k**-vector

Donde:

$$\begin{cases} m = (y_{max} - y_{min} + 2\delta\epsilon)/(n-1) \\ q = y_{min} - m - \delta\epsilon \end{cases} \tag{5}$$

Asi nuestro **k**-vetor debe cumplir $\mathbf{k}(1) = 0$ $\mathbf{k}(n) = n$ y además en la posición i-ésima $\mathbf{k}(i) = j$, donde j es el índice más grande que cumple:

$$\mathbf{s}(j) \leq \mathbf{y}(\mathbf{I}(i)) \tag{6}$$

Desde un punto de vista práctico los valores de $\mathbf{k}(i)$ representan el número de elementos de la base de datos $\mathbf{y}$ que están por debajo del valor $z(i)$.

Una vez que el **k**-vector ha sido creado podemos conocer cuantos y en que posición del vector $\mathbf{s}$ están los datos que buscamos ($[y_a, y_b]$). De hecho los índices asociados con estos valores en el vector $\mathbf{s}$ están dados por:

$$j_b = \lfloor \frac{y_a - q}{m} \rfloor \qquad \text{and} \qquad j_t = \lceil \frac{y_b - q}{m} \rceil \tag{7}$$

Donde $\lceil x \rceil$ representa al entero próximo inmediátamente superior y $\lfloor x \rfloor$ al entero próximo inmediátamente inferior. Una vez que tenemos los indices podemos obtener los indices de el rango de búsqueda de acuerdo a:

$$k_{start} = \mathbf{k}(j_b) + 1 \qquad \text{and} \qquad k_{end} = \mathbf{k}(j_t) \tag{8}$$

Asi finalmente los elementos buscados en la base de datos $\mathbf{y}(i) \in [y_a, y_b]$ son los elementos $\mathbf{y}(\mathbf{I}(k))$, donde $k \in [k_{start}, k_{end}]$. Sin embargo en la base de datos pueden aparecer elementos extraños originados por el hecho que en promedio los índices k_{star} y k_{end} tienen cada uno 50 % de probabilidades de no pertenecer a los intervalos $[y_a, z(j_a + 1)]$ y $[z(j_b, y_b)]$ respectivamente Asi si E_0 es la cantidad de elementos esperados en el rango $[z(j), z(j+1)]$ entonces el número de elementos extraños esperados en el rango $[y_a, y_b]$ es $\frac{2E_0}{2} = E_0 = \frac{n}{(n-1)} \approx 1$ para $n \gg 1$. Por ello en presencia de este elemento extraño, debemos hacer dos búsquedas locales más para eliminar dicho elemento. Formalmente:

$$\begin{cases} \text{verificar si} & \mathbf{y}(\mathbf{I}(\mathbf{k})) < y_a \\ \text{verificar si} & \mathbf{y}(\mathbf{I}(\mathbf{k})) > y_b \end{cases} \quad \text{Donde} \quad \begin{matrix} k = k_{start} \\ k = k_{end} \end{matrix} \tag{9}$$

Para base de datos grandes ($n \gg 1$), dado que $\lim_{n\to\infty} E_0 = 1$ el número de datos buscados puede ser aproximado por $n_d = k_{start} - k_{end}$, dado que el número esperado de elementos extraños se aproxima a $1 (E_0 \approx 1)$. De esta manera en promedio el método provee una solución con solo tres comparaciones y por lo tanto exibe una complejidad de $\mathcal{O}(3)$, el cual es independiente del tamaño de la base de datos[2].

2.1. Aplicaciones del método k-vector

2.1.1. Inversión de Funciones no-lineales

El método de **k**-vector exibe ventajas cuando es aplicado a bases de datos estáticas, por ello puede ser utilizado de manera eficiente para invertir funciones no-lineales de la forma $y = f(\mathbf{x})$, donde f es una función no lineal de n variables independientes.

Éste método no es el adecuado para invertir una función sólo una vez, ya que su eficiencia se ve reflejada cuando la inversión debe hacerse repetidas veces en algún rango de valores de la función. Una vez que el preprocesamiento se ha completado, la inversión puede ser creada y usada en cualquier momento. Otra restricción del método es para funciones que poseen singularidades, como por ejemplo la función Gamma ($\Gamma(x)$).

La metodología puede ser resumida en los siguientes diagramas de flujo. En el diagrama de la izquierda se resume el preprocesamiento y en el de la derecha la inversión misma.

[2]Este valor de complejidad se mantiene si el comportamiento del **k**-vector es lineal o cuasi-lineal con respecto a los indices

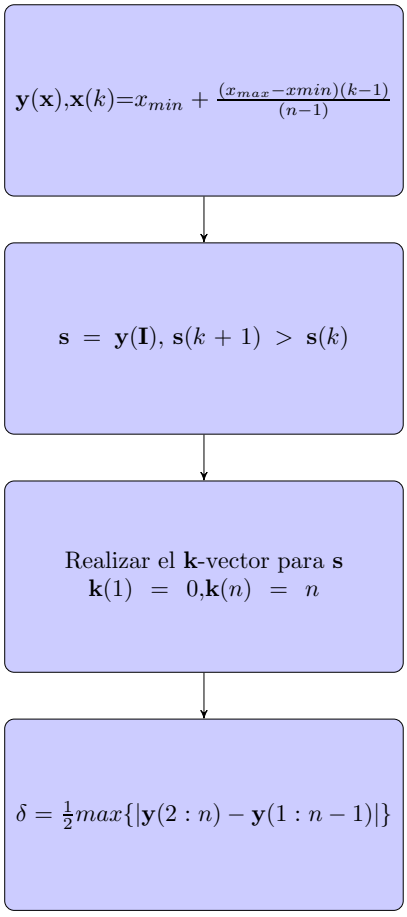

Figura 1: Preprocesamiento

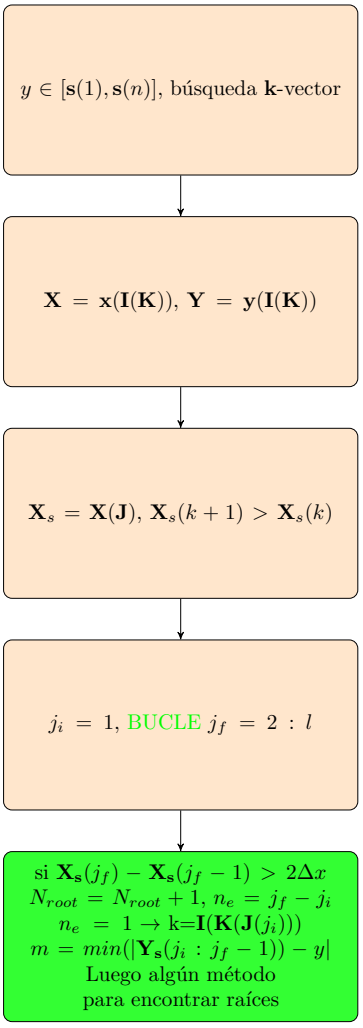

Figura 2: Inversión de una función no-lineal

2.1.2. Resolver sistemas de dos ecuaciones no-lineales n-dimensionales

Sea el siguiente sistema de dos ecuaciones:

$$y = f(\mathbf{x}) \qquad \text{and} \qquad y = g(\mathbf{x}) + c \tag{10}$$

Donde $f(\mathbf{x})$ y $g(\mathbf{x})$ pueden ser dos funciones nolineales dadas y c una constante cuyo valor puede cambiar. Se puede ver que el problema anterior es equivalente a resolver la siguiente ecuación:

$$c = f(\mathbf{x}) - g(\mathbf{x}) = h(\mathbf{x}) \tag{11}$$

Problema que se puede resolver con la metodología anterior.

2.1.3. Interpolación

Si consideramos una *look-up table* discreta de una dimensión con coordenadas $\mathbf{x} = \{x_1, x_2, x_3....x_n\}^T$ que mapea valores a otro vector $\mathbf{y} = \{y_1, y_2, y_3....y_n\}^T$. La intepolación con $\mathbf{k}$-vector es primero iniciada construyendo el $\mathbf{k}$-vector fuera del procesamiento en tiempo real. En este caso el $\mathbf{k}$-vector esta dado por $\mathbf{k} = \{k_1, k_2, k_3....k_n\}^T$ donde k_j es el número de valores de x que estan por debajo de la linea que une los puntos $[1, -\delta]$ $[n, x_n + \delta]$ al j-ésimo

indice. Para un dado tiempo de simulación el valor actual de x se denota como x^*. Entonces el indice del **k**-vector es calculado de acuerdo a:

$$j = floor(\frac{x^* - \delta - q}{m}) \tag{12}$$

Donde $m = \frac{x_n - x_1 + 2\delta}{(n-1)}$. Entonces, para calcular el valor interpolado y^*, se usa alguno de los métodos conocidos de interpolación (lineal, más alto orden, spline, spline cúbico..) para aproximar entre x_{k_j} y $x_{k_{j+1}}$.

3 Resultados

3.1. Raíces de funciones no-lineales

Realizamos una implementación del algoritmo descripto en (2)basados en el preprocesamiento y la búsqueda en si, mediante dos funciones que automatizan el método. Obtuvimos resultados muy buenos en varios sentidos comparados con los convencionales:

- El método halla **todas** las raíces en el rango que se lo prueba($[x_{min}, x_{max}]$)
- La velocidad de resolución casi no varía con el número de elementos de la base de datos
- El método encuentra solo algunas raíces cuando el número de elementos de la base de datos (la discretización de la función) no es lo suficientemente grande
- Una vez obtenido el **k**-vector para ese rango recalcular es casi inmediato
- El **k**-vector resulta una aproximación optima para el primer iterante de otros métodos de búsqueda de raíces(ej: Newton-Rapshon)

Como ejemplo de salida típica de la implementación para la función $y = sinc(x)$ en $[-5, 5]$ es

```
octave:3> nonlinear_inversion
Raiz:1 ---> -5
Raiz:2 ---> -4
Raiz:3 ---> -3
Raiz:4 ---> -2
Raiz:5 ---> -1
Raiz:6 ---> 1
Raiz:7 ---> 2
Raiz:8 ---> 3
Raiz:9 ---> 4
Raiz:10 ---> 5
elapsed_time =  0.033457
```

3.2. Intersección de dos funciones no-lineales

Utilizando las mismas funciones de la implementación anterior, realizamos un script que obtiene las intersecciones de dos funciones no-lineales, $sinc(t)$ y $sin(t)$ y el resultado que obtuvimos fue el siguiente:

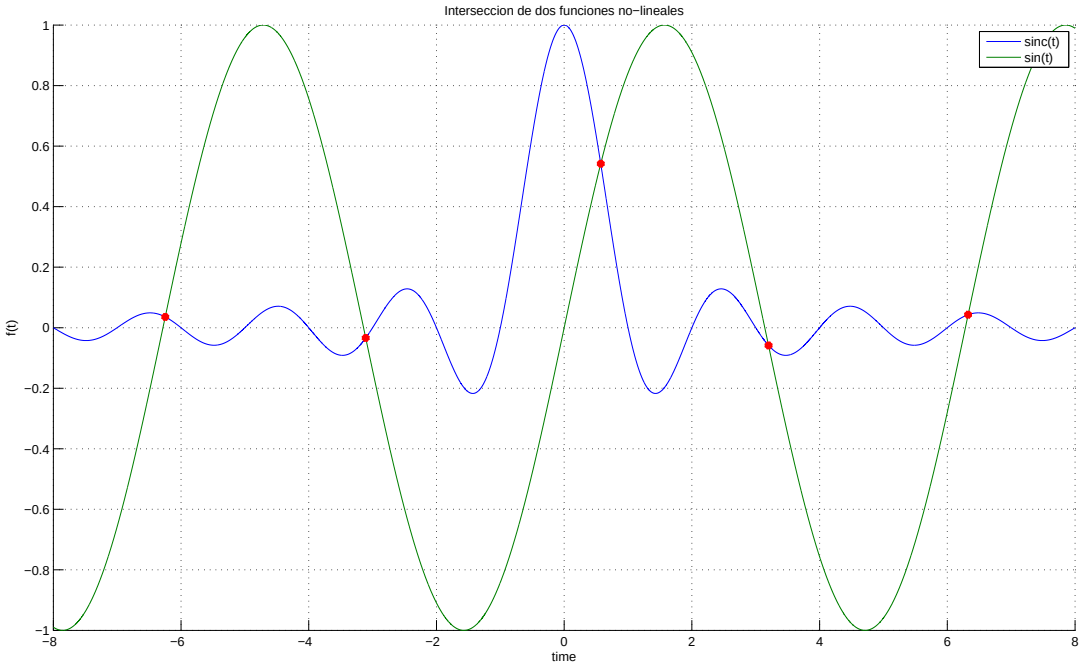

Figura 3: Solución a la intersección de dos funciones no-lineales

3.3. Interpolación

Para verificar que la magnitud de la base de datos no afecta considerablemente a la velocidad de resolución, realizamos un script que genera una base de datos variable y una búsqueda en particular con el método antes mencionado, a cada búsqueda es tomado el tiempo de resolución y luego se grafica Los resultados fueron:

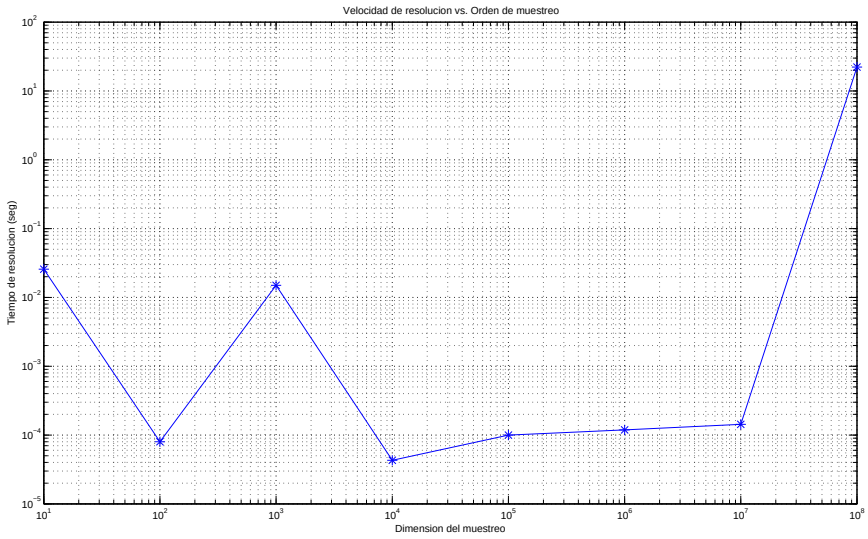

Figura 4: Velocidad de resolución vs orden de la base de datos

4 conclusiones

4.1. Raíces de funciones no-lineales

Al observar los resultados obtenidos del tratamiento de la función $sinc(x)$ con el método diagramado a partir del **k**-vector, se puede afirmar que es indiscutible la efectividad de dicho método comparado con otras metodologías para encontrar raíces en un determinado intervalo. Ésto se debe a que determinadas búsquedas de raíces, como ser el método de Newton-Raphson sin ser optimizado por el **k**-vector, encuentran únicamente una raíz de todas las posibles en dicho intervalo con una gran cantidad de iteraciones. Por otra parte este método posee una gran precisión y obtiene todas las raíces de la función en el intervalo solicitado. A su vez, el tiempo de resolución del mismo es considerablemente pequeño al compararlo con los tiempos de resolución de otros métodos, como ser el método de la secante. Éste último es un método eficaz, si se busca una sola raíz, pero lento en su aplicación. El pequeño tiempo de resolución es debido a la implemetación del método de búsqueda del k-vector, quien otorga un iterante inicial próximo a cada raíz, para luego utilizar el método de Newton-Raphson con dicho iterante inicial. De ésta manera, el método de Newton-Raphson encuentra a las raíces con 2 o, a lo sumo, 3 iteraciones debido al óptimo iterante inicial dado por la utilización del método del **k**-vector.

4.2. Intersección de dos funciones no-lineales

Como se puede observar en la figura 3, es factible encontrar los puntos de intersección de dos funciones no-lineales bajo el método **k**-vector. Éste proceso lleva un leve costo en cuanto a la programación, debido a que es posible implementarlo directamente utilizando la búsqueda de raíces de funciones no-lineales: Al restar las funciones entre si, se genera una nueva función, y se le hallan a ésta sus ceros. Éstos ceros van a representar las coordenada de las intersecciones de las funciones originales. Como se menciono con anterioridad, el método de búsqueda de raíces es muy efectivo tanto en tiempo de resolución como en precisión. Al ser el método para encontrar las intersecciones de dos funciones una variante del método anterior, éste posee las mismas características.

4.3. Interpolación

Al observar la Figura (4) se puede ver que el método de interpolación utilizado es muy efectivo, no solo cuando se necesita un método de resolución rápido, sino también para situaciones en las cuales el problema conlleva una gran serie de datos independientes, obtenidos empíricamente. Éste tipo de metodología es recomendable cuando se enfrenta a situaciones en las cuales la base de datos cambia continuamente, ya que el método posée un tiempo de resolución muy inferior a otros métodos cuando las bases de datos muestreados es considerablemente grande. Éste tipo de base de datos no afecta al método del **k**-vector en términos de eficiencia ni de velocidad, como puede observarse en la figura (4).

Referencias

[1] A **k**-vector approach to sampling, interpolation, and aproximation. Daniele Mortari and Jonathan Rogers

[2] **k**-vector Range searching techniques. Daniele Mortari, Beny Neta